VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Rozšíření MQTT brokeru o archivaci a vizualizaci dat

Extension of MQTT Broker to Archive and Visualize Data

2018 Bc. Kevin Kiedroň

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Kevin Kiedroň**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Rozšíření MQTT brokeru o archivaci a vizualizaci dat
Extension of MQTT Broker to Archive and Visualize Data

Jazyk vypracování: čeština

Zásady pro vypracování:

Vytvořte systém, který rozšíří vlastnosti MQTT Brokeru o možnost archivace dat. Rozšíření koncipujte tak, aby bylo jednoduše aplikovatelné na kterýkoli MQTT Broker. V systému bude možné nastavit která témata budou archivována a také bude možné nastavovat limity pro archivaci, jako například hloubka historie, maximální velikost archivovaných zpráv a podobně. Systém bude také umožňovat jednoduchou vizualizaci archivovaných dat ve formě generovaných grafů a také bude možné provádět nad archivovanými daty základní statistické operace jako je maximální, minimální nebo průměrná hodnota za určené časové období.

1. Stručně popište protokol MQTT a jeho vlastnosti.
2. Navrhněte a naprogramujte program pro archivaci dat přenášených přes MQTT Broker.
3. Navrhněte a implementujte vhodné rozhraní, kterým si bude uživatel moci archivní data získat.
4. Do systému přidejte možnost získání jednoduchých grafů a základních statistických údajů jako je průměrná hodnota, regrese, predikce hodnoty, maximální a minimální hodnota a případně další.
5. Proveďte zátěžové testy a testy stability systému.

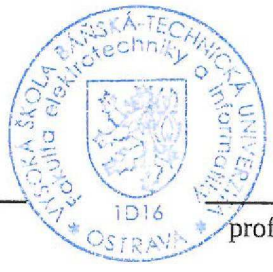
Seznam doporučené odborné literatury:

[1] Mosquitto project, online: https://mosquitto.org/
[2] MQTT Version 3.1.1, dokumentace protokolu, online: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. David Seidl, Ph.D.**

Datum zadání: 01.09.2017
Datum odevzdání: 30.04.2018



doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 30. duben 2018

.....

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava.

V Ostravě 30. duben 2018

Děkuji panu Ing. Davidu Seidlovi, Ph.D., za poskytnutí zajímavého tématu a za pomoc a cenné rady, které mi při zpracovávání této práce poskytl.

Abstrakt

V následujícím textu se budu věnovat rozšíření MQTT brokerů, které umožnuje sběr dat a jejich následnou vizualizaci. Zprvu lehce přiblížím MQTT technologii jako takovou, pro čtenáře neznalé problematiky protokolu MQTT. Následně se budu zabývat procesem vývoje tohoto rozšíření, přiblížím použité nástroje a zdůvodním jejich použití, případně použité nástroje porovnám s nepoužitými alternativami. Poté přiblížím návrh softwaru a popíšu databázovou úroveň programu a rovněž to, jak k databázovaným datům přistupovat. Ve finále se budu věnovat záťežovým testům a testům stability celého řešení.

Klíčová slova: MQTT, diplomová práce, archivace dat, vizualizace dat, IoT

Abstract

In the following text, I will focus on extension for MQTT brokers that collects data and provides their visualization. First of all MQTT protocol will be introduced to the reader. After that process of development will be described. I will introduce used technologies and tools and I will also specify why these were used instead of alternative solutions. Then design of the software will be described and database layer will be introduced to the reader. There will also be a part where interfaces for data access will be described. Final part of the text will be about stability and stress tests of entire solution.

Key Words: MQTT, thesis, data collecting, data visualization, IoT

Obsah

Seznam použitých zkratek a symbolů

IoT	–	Internet of Things
PUBACK	–	Publication Acknowledgement
PUBREL	–	Publish release
PUBCOMP	–	Publish complete
PUBREC	–	Publish received
SQL	–	Structured Query Language
TCP	–	Transmission Control Protocol
UDP	–	User Datagram Protocol
API	–	Application Programming Interface
QoS	–	Quality of Service
TLS	–	Protokol Transport Layer Security
ACL	–	Access Control List
HTTP	–	Hypertext Transfer Protocol
REST	–	Representational State Transfer
CoAP	–	Constrained Application Protocol
OMG	–	Object Management Group
DDS	–	Data Distribution Service
M2M	–	Machine-to-Machine
HPC	–	High-performance computing
WAN	–	Wide Area Network
LAN	–	Local Area Network
GPL	–	General Public License
ORM	–	Object-relational mapping
JSON	–	JavaScript Object Notation
SŘBD	–	Systém řízení báze dat
URL	–	Uniform Resource Locator

Seznam obrázků

Seznam tabulek

1 Úvod

V současné době se velice rozmohla zařízení spadající do internetu věcí (dále pouze IoT). Pokud hovoříme o IoT, máme na mysli scénáře, kdy se vlastnosti síťového připojení a schopnosti provádět výpočty rozšiřují na objekty, čidla či předměty každodenní potřeby, které normálně za počítače považovány nejsou. Tato zařízení jsou poté schopna generovat, vyměňovat nebo konzumovat data jen s minimem lidské interakce. Tuto definici nelze považovat za univerzální definici pokrývající celou problematiku IoT, pro potřeby tohoto textu je však dostačující.

Koncept kombinování počítačů, čidel a sítí k monitorování a kontrolování zařízení existuje již několik desetiletí, ale teprve před pár lety se vliv několika technologií zasloužil o rozšíření povědomí o IoT mezi širokou veřejností.

Různé IoT implementace používají různé komunikační modely, každý má své vlastní charakteristiky. Tato práce se věnuje technologii MQTT, která se zaměřuje na komunikaci mezi zařízeními v sítích, které disponují převážně různými senzory a tzv. embedded zařízeními. Konkrétně se budu zabývat implementaci programu pro sběr dat pomocí této technologie.

Cílem práce je tedy seznámit čtenáře s použitými technologiemi, následně navrhnout a naprogramovat program, jehož klíčovou funkcí bude sběr a následná archivace dat přenášených přes MQTT broker. Rovněž se v textu budu zabývat návrhem a implementací vhodného rozhraní pro přístup k archivovaným datům. Poté se zaměřím na implementaci některých základních funkcí, které se nad danými daty dají provádět (např. zjištění minima, maxima atp.). Závěr práce bude věnován testům stability systému a následnému zhodnocení celého snažení.

2 Technologie MQTT

MQTT je jednoduchý a nenáročný protokol pro posílání zpráv. MQTT byl vyvinut společností IBM, v současné době o protokol pečuje Eclipse foundation. Přenos dat probíhá pomoci TCP protokolu a používá návrhový vzor publisher - subscriber (tedy vzor založený na publikování a odběru zpráv). Při práci s MQTT rovněž narazíme na pojem broker. Broker je centrální bod, který se stará o výměnu zpráv. Samotný obsah zpráv není nijak daný a záleží na uživateli, jak jej definuje. Velikost zprávy je limitována 256 MB, což je pro většinu případů užití naprosto dostačující [1]. Maximální velikost zpráv lze překročit, větší velikost zpráv však musí být podporována na straně brokeru [2].

V následujích řádcích budou stručně představeny některé stěžejní funkce protokolu a protokol samotný.

2.1 Publikování a odběr zpráv

Jak již bylo zmíněno výše, funkcionalita MQTT protokolu je založena na principu publikování zpráv a odběru témat. Několik klientů se připojí na broker a začne odebírat témata, o která mají zájem. Klienti, kteří jsou také připojeni k danému brokeru, mohou publikovat zprávy do těchto témat. Mnoho klientů může odebírat stejná témata a dale zpracovat získaná data podle vlastních potřeb. Lze tedy například velmi jednoduše implementovat program, který získaná data uloží do databáze, nebo je například pomocí API odešle na Twitter, kde podá informaci o aktuálním měření.

2.2 Témata a odběry

Zprávy jsou v rámci MQTT protokolu tedy publikovány do témat. Témata není třeba nijak konfigurovat. Přistupuje se k nim hierarchicky, lomítko (/) se používá jako oddělovač. To umožňuje rozumné uspořádání společných témat, která mají být vytvořena (podobně jako u klasických souborových systémů). Jako příklad se často uvádí případ, kdy počítače publikují informaci o teplotě pevných disků do témat, která jsou navržena podle této šablony (COMPUTER_NAME a HARDDRIVE_NAME budou definovány konkrétním případem):

```
sensors/COMPUTER_NAME/temperature/HARDDRIVE_NAME
```

Klienti se mohou přihlásit buďto přímo ke konkrétnímu tématu, nebo mohou využít tzv. wildcards, které jsou k dispozici rovnou dvě (+ nebo #).

Wildcard + je vhodné použít v případě, kdy klienta zajímají všechna témata na dané úrovni hierarchie. Pokud ale klienta zajímají všechna zbylá témata na ostatních úrovních dané hierarchie, použije wildcard #.

Příkladem může být situace, kdy je potřeba sbírat informace ze všech senzorů z určitého počítače a je tedy potřeba nadefinovat odběr tématu následovně:

```
sensors/COMPUTER_NAME/#
```

Pokud je ale potřeba sbírat ze všech monitorovaných komponent, například pouze data o teplotě, je nutno nadefinovat odběr tématu takto:

```
sensors/COMPUTER_NAME/temperature/+
```

V MQTT jsou rovněž speciální témata, na která jsou posílána systémová data, týkající se daného brokeru. Tato témata jsou začínající řetězcem `$SYS` a ke konkrétním datům se přistupuje stejně jako v předešlých případech hierarchicky. Pokud tedy například uživatel požaduje počet aktuálně připojených klientů k brokeru začne odebírat následující téma:

```
$SYS/broker/clients/total
```

2.3 Kvalita služby

MQTT definuje tři úrovně kvality služby (QoS - Quality of Service). QoS určuje, jak moc bude broker/klient usilovat o to, aby byla zpráva korektně doručena. QoS se definuje při odesílání zprávy a může být libovolně zvolen. Klient může rovněž zvolit, s jakou QoS úrovní bude témata odebírat. To znamená, že klient nastaví maximální QoS hodnotu a na základě toho bude ovlivněn odběr zpráv.

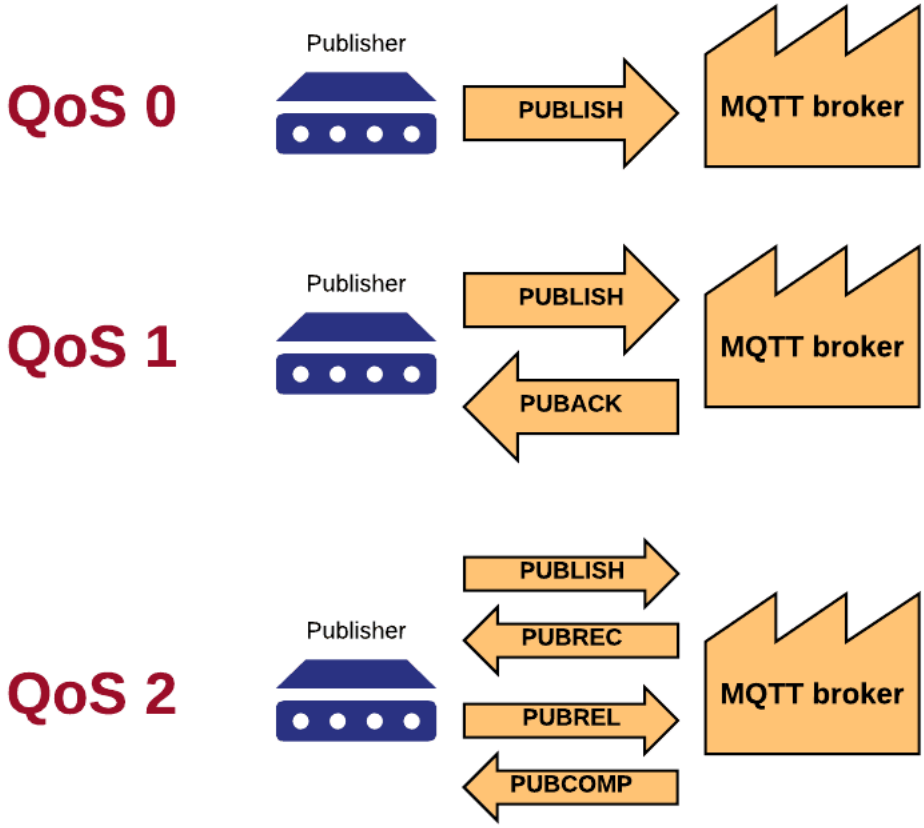
Pokud klient publikuje zprávu s úrovní QoS nastavenou na 0, pošle zprávu na broker bez další režie a o nic se nestará; pokud zpráva na broker korektně dorazí, broker zprávu přepošle odběratelům daného tématu stejným způsobem.

Na úrovni QoS nastavenou na hodnotu 1 pošle klient zprávu typu PUBLISH na broker a čeká. Broker tuto zprávu přijme a pošle ji odběratelům, zpráva je nastavená na QoS hodnotu 1, nebo, v případě, že zařízení zprávy QoS 1 nepřijmá, je nastavena na hodnotu 0. Obecně však platí, že broker odešle zprávu se stejnou QoS hodnotou s jakou ji přijme. V případě, kdy je QoS nastaven na 1, odesílatel čeká na potvrzení od příjemců. Příjemce, který zprávu přijal, pošle potvrzení (zpráva typu PUBACK) na broker. Ten zprávu smaže a informuje odesílatele o tom, že zpráva prošla přes broker (opět zprávou typu PUBACK). Odesílatel na základě obdržené informace zprávu zahodí.

Poslední možná úroveň QoS je úroveň 2. Pokud je zpráva odeslána s touto hodnotou, je stejně jako v předchozím případě zpráva typu PUBLISH odeslána na broker, který ji rozdistribuje mezi odběratele a odešle odesílateli zprávu typu PUBREC (tedy potvrzení přijetí). Odesílatel na to zareaguje zprávou PUBREL, na základě které broker zprávu smaže, následně odešle zprávu typu PUBCOMP, čímž odesílateli oznámí, že je proces publikace u konce a výměna je ukončena.

Čím vyšší je úroveň QoS, tím vyšší je spolehlivost doručení zprávy. To je bohužel vykoupeno většími nároky na režii a přenos informace přes síť. Výše uvedené informace o chování QoS

jsou obecné informace z dokumentace; implementace QoSu se může v některých specifických případech lehce lišit v závislosti na potřebách konkrétního případu.



Obrázek 1: Rozdíly v komunikaci za použití různých úrovní QoS [1]

2.4 Zachycené zprávy (Retained Messages)

Zachycené zprávy, neboli Retained Messages, jsou zprávy, které broker uchová i potom, co byly odeslány všem odběratelům. Pokud se objeví nový odběratel daného témata, zachycená zpráva bude odeslána i jemu. Jakákoliv zpráva může být nastavená jako Retained Message.

2.5 Poslední vůle (Wills)

V okamžiku, kdy se klient připojí k brokeru, informuje broker o tom, že má připravenou „poslední vůli“ (Will). Tím je myšlena zpráva, která se odešle, pokud se klient neočekávaně odpojí. Zpráva poslední vůle má, stejně jako ostatní zprávy, nastavené téma, QoS a může být rovněž zachycena.

2.6 Čistá relace a odolná konektivita

Klient může dané spojení nastavit jako „Čistou relaci“ (Clean session), což je rovněž známo jako „čistý start“. Pokud klient nenastaví relaci jako čistou, je navázané spojení chápáno jako odolné. To znamená, že pokud se klient odpojí, všechny jeho odběry zůstanou a všechny zprávy s QoS hodnotou 1 nebo 2 budou uloženy, dokud se v budoucnu znovu nepřipojí. Pokud je spojení nastaveno jako čistá relace, všechny odběry budou odstraněny v okamžiku odpojení klienta.

MQTT protokol je tedy mocný nástroj, je velmi snadno použitelný, má minimální nároky na hardware a dá se velice snadno přizpůsobit různým případům užití. Existuje také spousta klientských knihoven pro velké množství programovacích jazyků (Java, Python, JavaScript, Ruby, Go, atd.). Rovněž existuje varianta protokolu MQTT zvaná MQTT-SN (MQTT for Sensor Networks), která je vhodná pro non-TCP spojení (například technologie ZigBee, iQRF). Protokol bývá velmi často používán nadšenci pro moderní technologie, ale narazíme na něj rovněž ve velkých komerčních projektech. Asi nejznámější aplikací protokolu MQTT je Messenger společnosti Facebook, který těží z nízkých nároků na síť, což je pro mobilní komunikátor klíčovou vlastností.

Pro vývoj programu je tedy potřeba řešení, které pokryje výše zmíněné funkcionality protokolu MQTT. Takové řešení není nutno vyvíjet, jelikož ho stejně jako samotný MQTT protokol nabízí Eclipse foundation. Jedná se o Eclipse Paho, což je open-source implementace klientské části. Knihovna Eclipse Paho je dostupná pro většinu programovacích jazyků a disponuje implementacemi všech potřebných metod. Pro zprovoznění brokeru a testovací účely využiji Eclipse Mosquitto, což je, stejně jako v předchozích případech, open-source program vyvinutý Eclipse foundation. Eclipse Mosquitto je broker, který doplní dva populární programy mosquitto_pub a mosquitto_sub, které jsou součástí stejného projektu a slouží k odběru a publikování zpráv.

2.7 Zabezpečení protokolu MQTT

Jedním ze způsobů, jak zabezpečit komunikaci mezi zařízeními a brokerem je použití autentizace. Jedná se o zabezpečení na úrovni aplikační a transportní vrstvy. Na úrovni transportní vrstvy, může být použit protokol TLS, který zaručí autentizaci klienta vůči serveru pomocí certifikátů. Autentizace může probíhat také na aplikační vrstvě, kdy je serveru za pomocí protokolu MQTT poskytnuto uživatelské jméno a heslo. Nejznámější implementace brokerů jako jsou například Mosquitto nebo Mosca touto formou zabezpečení disponují. Ve výchozím stavu je autentizace pro broker vypnutá.

Důležitou informací je, že uživatelské jméno a heslo použité pro autentizaci, jsou po síti přenášeny v nezašifrované formě. Tyto údaje tak mohou být zachyceny útočníkem a zneužity. Jediným způsobem, jak zaručit úplné zabezpečení přenosu uživatelského jména a hesla, je použití šifrovaného přenosu.

Zabezpečení lze dále rozšířit autorizací. Nejjednodušší autorizační metodou je použití ACL souboru, v něm je definováno, do jakých témat můžou konkrétní uživatelé publikovat, případně

která tématu mohou tito uživatelé odebírat, uživatel se však musí k brokeru přihlásit pod přihlašovacím jménem, které je uvedeno v ACL souboru [4].

2.8 Propojení MQTT brokerů

MQTT brokery je mezi sebou možné propojit. To znamená, že je možné zařídit, aby mezi sebou bylo hierarchicky propojeno více MQTT brokerů, kdy jsou zprávy směrovány mezi jednotlivými brokery standartním MQTT mechanismem. Tato funkcionalita se většinou používá, pro sdílení zpráv mezi jednotlivými systémy. Zjednodušeně lze říct, že se z brokeru, přijímajícího zprávy z jiného brokeru, stává plnohodnotný MQTT klient.

Pokud je tedy žádoucí, aby broker přijímal data jiného brokeru, je nutné, nastavit mezi brokery bridge (most). Na brokeru se tedy nastaví témata, která se budou přeposílat na další broker podle prefixu řetězce označující téma, případně jak se přeloží tento řetězec v rámci vzdáleného brokeru. Toto nastavení může fungovat oběma směry, je tedy možné posílat zprávy vzdálenému brokeru a stejně tak zprávy ze vzdáleného brokeru přijímat [5].

2.9 MQTT Alternativy

V dobách, kdy IoT začínalo a začalo se přemýšlet, prostřednictvím jakého protokolu by zařízení mohla komunikovat s okolním světem, bylo snahou využít již existující a dobře známé internetové protokoly.

Mezi protokoly, které by mohly být použity, patří nepochybně protokoly TCP a UDP. Z principu funkcionality těchto protokolů je patrné, že spojově orientovaný TCP protokol bude pomalejší než nespojově orientovaný protokol UDP; na druhou stranu je však TCP spolehlivý protokol umožňující opakované odeslání ztracených packetů, dokud nebude jasné, že packety byly příjemcem přijaty. Protokol UDP je velmi jednoduchý na implementaci a představuje minimální zátěž pro zařízení, použití UDP však s sebou nese i některá úskalí. Protokol UDP je vhodný především pro jednosměrné odesílání dat od klienta k serveru. Opačný komunikační kanál vyžaduje, aby server znal adresy zařízení a zařízení v pravidelných intervalech naslouchalo zprávám ze serveru, nebo by klient musel používat pull přístup a data si ze serveru pravidelně stahovat.

UDP protokol je vhodné použít v aplikacích, u nichž bude tolerována ztráta dat během přenosu, tedy například v real-time systémech, kde je upřednostněna rychlost [7].

Při výběru protokolu je tedy nutno zvážit, jaké jsou kladeny požadavky na aplikaci. Faktem ale je, že pokud bude zvolen jeden z těchto protokolů, bude nutností implementovat vhodný systém pro klientské zařízení a server tak, aby komunikace odpovídala požadavkům. Snažším řešením může být volba protokolu vyšší úrovně nebo rovnou protokolu, určeného pro komunikaci IoT zařízení.

Je například možné uvažovat scénář, kdy bude použit protokol HTTP v kombinaci s REST API. Kombinace těchto technologií s sebou však nese několik nevýhod. Prokol HTTP při přenosu

informací posílá relativně velké množství provozních dat, která přenos umožňují. Implementace protokolu HTTP je rovněž netriviální a protokol HTTP také nedisponuje různými úrovněmi QoS. Vývoj systému, který by mohl pracovat s těmito technologiemi, by byl relativně náročný - musela by být vyvinuta aplikace, která by byla spuštěna na serveru a předávala by například data mezi klienty.

Od IoT zařízení se neočekává vysoký výkon, velká paměť nebo dobrá konektivita, je tedy patrné, že kombinace technologií HTTP a REST není pro tento typ zařízení vhodná.

Proto vznikl protokol CoAP, který je na rozdíl od protokolu HTTP binární a místo TCP protokolu používá protokol UDP. Rovněž v sobě zahrnuje QoS. Protokol CoAP nepřenáší tolik provozních dat, jelikož CoAP zprávy neobsahují textové hlavičky, ale binární příznaky. Implementace je rovněž jednodušší. Práce s protokolem CoAP se velmi podobá práci s protokolem HTTP. Výhodou CoAP tedy je, že byl narozdíl od HTTP, přímo navržen pro IoT, je tedy dostatečně lehký [3].

2.9.1 Rozdíly protokolu CoAP a MQTT

Protokol CoAP je primárně protokol, který zprostředkovává komunikaci mezi klientem a serverem. Protokol CoAP je vhodné použít v případech, kdy je požadováno přenášet informace o stavu, nikoliv provádět přenos dat na základě událostí.

Protokol MQTT zprostředkovává komunikaci mnoha klientů prostřednictvím cetrálního serveru (brokeru). Nedělá rozdíly mezi producentem dat a spotřebitelem tím, že nechá klienty publikovat data a broker rozhoduje, kam směrovat a kopírovat zprávy. Ačkoliv je v protokolu MQTT implementována podpora pro perzistentní data, MQTT nejlépe funguje jako komunikační sběrnice pro živá data, tedy neustále se měnící data.

MQTT neposkytuje podporu pro označování zpráv s typy nebo jinými metadaty, které by klientům pomohly pochopit. MQTT zprávy mohou být použity pro jakýkoli účel, ale všichni klienti musí dopředu znát formáty zpráv, aby je následně mohli zpracovat.

CoAP naopak poskytuje vestavěnou podporu pro vyjednávání a zjišťování obsahu zpráv, umožňující zařízením navzájem prozkoumávat způsoby výměny dat [6].

Ze zmíněných rozdílů je patrné, že protokoly MQTT a CoAP není vhodné porovnávat, oba protokoly totiž fungují na jiném principu. Volba protokolu by měla záviset na konkrétním případu užití.

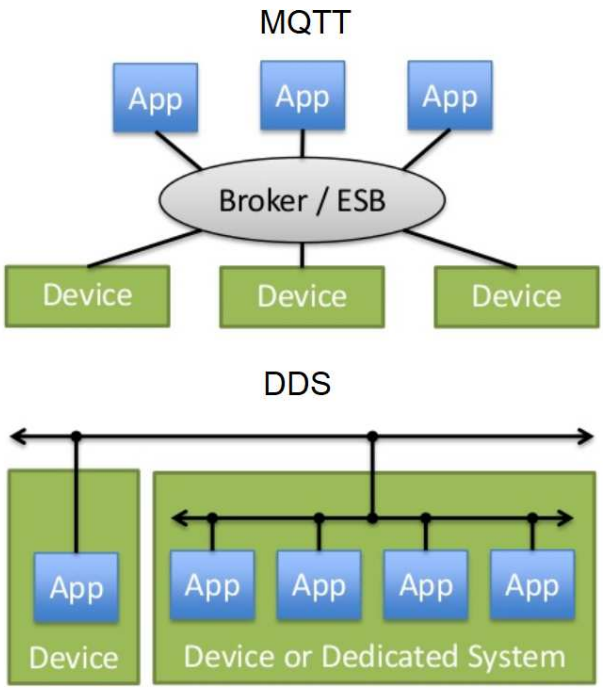
2.9.2 Protokoly MQTT a DDS

Mnohem vhodnější kombinací protokolů pro porovnání jsou protokoly MQTT a DDS.

DDS je middleware protokol vyvinutý Object Management Group (OMG) konsorciem. K dispozici je jak komerční, tak open source verze. Vývojáři mají k dispozici knihovny a moduly pro programovací jazyky Ada, C, C++, C#, Java, Scala, Lua, Pharo a Ruby.

V obou případech se jedná o protokoly založené na vzoru publisher - subscriber a v obou případech se jedná o protokoly navržené speciálně pro komunikaci machine-to-machine (M2M), což je základem pro IoT.

MQTT je optimalizován pro centralizovaný sběr a analýzu dat - připojení senzorů a mobilních zařízení k aplikacím spuštěným v datovém centru. Na druhou stranu protokol DDS je optimalizován pro distribuované zpracovávání - vzájemně přímo propojené senzory, zařízení a aplikace bez jakékoliv závislosti na centralizované IT infrastruktuře. Zjednodušeně řečeno, rozdíly mezi MQTT a DDS se projevují v jejich základních architekturách (Obrázek 2).



Obrázek 2: Rozdílná architektura protokolů MQTT a DSS [10]

V případě protokolu MQTT komunikace probíhá prostřednictvím centrálního brokeru. Protokol DDS je decentralizován. Objekty, které generují data, komunikují přímo s aplikacemi a zařízeními, které data konzumují. Zjednodušeně by se tento typ komunikace dal označit jako peer-to-peer. Data přicházejí do datového centra, pouze pokud je to vyžádáno.

MQTT protokol je vhodný pro klasické M2M aplikace, ve kterých klientský počítač hovoří se serverovým zařízením one-to-one. Příkladem mohou být senzory, které monitorují ropné vrty a potrubí. Protokol DDS je nejvhodnější použít, pokud není veškeré zpracovávání dat centralizováno. Uvažujme například systém monitorování pacienta. Data senzorů (například monitorování životních funkcí) jsou vizualizována na monitoru u lůžka, respektive na stanici zdravotní sestry, pro elektronické zdravotní záznamy a dokonce i na mobilním zařízení lékaře. Bylo by velmi neefektivní směrovat data snímače do výše zmíněných zařízení, prostřednictvím datového centra. Může to být dokonce technicky neudržitelné kvůli souhrnnému požadavku na šířku pásma [9].

	MQTT	**DDS**
Síť	Obvykle WAN, někdy LAN	Kombinace sdílené paměti, sběrnice, LAN, WAN
Administrace	Možno provádět změny konfigurací, počítá se s dostupností technika	Obvykle autonomní, technik nemusí být vždy dostupný
Rychlost odesílání zpráv	Několik zpráv za sekundu z jednoho zařízení	Několik desetitisíců zpráv za sekundu z jednoho zařízení
Latence	Stovky milisekund až sekundy	Stovky mikrosekund až milisekundy
Odolnost proti chybám	Lze tolerovat krátké přerušení služby	Krátké přerušení služby může mít katastrofické následky

Tabulka 1: Rozdílné nároky na protokoly v rámci aplikací [10]

Oba protokoly tedy poskytují standartní komunikační základy pro IoT; jejich architektura je ale odlišná a odlišné jsou tedy i aplikace, ve kterých jsou oba protokoly používány. DDS protokol bývá používán v nemocničním vybavení, HPC nebo k řízení robotů či vozidel bez lidské posádky. MQTT protokol je vhodný například pro monitorování spotřeby elektrické energie. Je tedy nutné zvážit nároky na protokol a následně jej vhodně zvolit. Tabulka 1 obsahuje přehled nároků, které jsou kladeny na protokoly MQTT a DDS v rámci různých aplikací.

3 Použité technologie a jejich alternativy

Při výběru vhodných technologií pro vývoj výše zmíněného programu bylo snahou využít otevřený software (open-source software). Dalším kritériem bylo zvolit takové technologie, které umožňují rychlý a flexibilní vývoj. Samozřejmostí je kompatibilita zvoleného softwaru se všemi hlavními operačními systémy a platformami.

3.1 Databáze

Pro potřebu archivace dat je zapotřebí vhodná databáze. Jelikož mám bohaté zkušenosti s SQL databázemi, zvolil jsem relační databázi MySQL, konkrétně MySQL Community Edition ve verzi 5.7.21, která spadá pod licenci GPL, a je tedy vhodná k použití pro vývoj výše zmíněného programu. Systém řízení báze dat (dále jen SŘBD) MySQL byl zvolen rovněž kvůli detailní dokumentaci a kvalitní komunitní podpoře.

Požadavkem, který byl z mé strany kladen na databázi, byla podpora uložených procedur (stored procedures) a triggerů; to je jedním z důvodů, proč jsem nesáhl po některé z NoSQL databázi, které nedisponují ekvivalentní náhradou za uložené procedury a nelze v nich používat standardizovaný strukturovaný dotazovací jazyk. Absence těchto funkcionalit by zpomalila vývoj a část funkcionalit databázové vrstvy by se musela přesunout na vrstvu aplikační, která by nebyla natolik výkonná.

Právě zmíněné NoSQL databáze by měly být nejvhodnější volbou pro databázi obsahující typ dat sbíraných v tomto projektu. NoSQL databáze se hojně používají v oblastech, kde se pracuje s velkým množstvím dat (big data), dále se pak vyznačují jednoduchostí návrhu a také tím, že jsou oproštěny od relačního modelu. Pro tento konkrétní případ užití by byla vhodná například InfluxDB nebo jiná time-series databáze. Tyto databáze se orientují na tzv. time-series data, tedy data závislá na čase, a disponují nástroji pro efektivní práci nad těmito daty. Použitím takovéto databáze by bylo možné dosáhnout vysoké míry optimalizace na databázové úrovni.

Z výše zmíněných důvodů a také z důvodu nekompatibilního ORM, o kterém se zmiňuji, nebude výsledný program NoSQL databáze podporovat.

3.2 Programovací jazyk

Pro vývoj programu jsem zvolil skriptovací programovací jazyk Python ve verzi 3.6. Důvodem pro volbu Pythonu byla kompatibilita se všemi hlavními platformami, velké množství modulů a otevřená licenční politika. Python rovněž umožňuje velice rychlý a flexibilní vývoj.

Mezi další programovací jazyky, které připadaly v úvahu a splňovaly otevřenou licenční politiku a kompatibilitu s různými systémy, patří Java a JavaScript, pro tyto jazyky je totiž dostupná oficiální MQTT klientská knihovna a programy napsané v těchto jazycích není potřeba kompilovat pro konkrétní operační systém.

Java nebyla použita, jelikož je pro potřeby implementace systému pro sběr dat zbytečně velká a robustní. Rovněž vývoj v programovacím jazyku Java není natolik flexibilní a rychlý, jedním z důvodů je, že disponuje silnou typovou kontrolou.

JavaScript nebyl vybrán jelikož oficiální klientská MQTT knihovna pro tento programovací jazyk používá pouze WebSockety, takže je možné napsaný program používat, pouze pokud má broker povoleno používání WebSocketů. Existuje rovněž komunitou vyvíjená verze pro Node.js, která umožňuje standardní TCP spojení, zde ale hrozí velmi omezená podpora, slabá dokumentace nebo jiné problémy.

3.2.1 Použité moduly

V několika následujících odstavcích představím některé z použitých modulů a zaměřím se především na ty, které v programu hrají velkou roli a nejsou součástí základní instalace Pythonu.

Pro rychlou a pohodlnou práci s databází je vhodné použít některou z ORM knihoven, tedy knihovnu zabývající se metodou mapování relační databáze na objekty. SQLAlchemy je právě takovou knihovnou, která mimo to disponuje celou řadou nástrojů pro práci s databází. SQLAlchemy podporuje velké množství databázových systémů, například SQLite, Postgresql, MySQL, MS-SQL nebo Firebird. Z tohoto faktu vyplývá, že výsledný program nebude závislý na specifickém systému řízení báze dat, jelikož databázová vrstva aplikace přeloží požadavky pro daný databázový systém. Tento modul jsem zvolil, protože se jedná o nejrozšířenější ORM modul pro Python s dobrou podporou a je rovněž úzce propojen s knihovnou Flask.

Vzájemná kompatibilita modulů SQLAlchemy a Flask je příhodná, modul Flask je totiž microframework, který úmožňuje vytvořit rychle a jednoduše kvalitní REST API pro přístup k archivovaným datům. Flask je tedy také používán.

Jelikož zadání vyžaduje nějaké z komplexnějších statistkých operací a datových analýz nad nasbíranými daty (např. lineární regresi nebo predikci), je příhodné sáhnout po některých modulech zaměřených na práci s daty a jejich zpracování. Tím modulem je scikit-learn. Tento modul se dá jednoduše charakterizovat jako sada nástrojů pro strojové učení, data mining a analýzu dat. Scikit-learn je založen na modulech SciPy, NumPy a matplotlib. Poslední dva jmenované je také nutno představit.

NumPy je modul pro vědecké výpočty v Pythonu. Mezi jeho hlavní přednosti patří například implementace N-dimenzionálního pole, nástroje pro integraci C/C++ a Fortran kódu a velká podpora pro lineární algebru. Dalším, v projektu hojně používaným modulem, je modul zvaný Matplotlib. Jak již název napovídá, jedná se o modul zabývající se generováním grafů a vzhledem k zaměření programu dává smysl jej použít. Modul se používá velmi dobře a disponuje širokými možnostmi pro vykreslení různých typů grafů.

Posledním modulem, který je nutno alespoň velmi stručně představit, je modul Pandas. Pandas je velmi výkonná knihovna pro datové analýzy, disponující celou řadou datových struktur, které jsou vhodné pro datové analýzy, a operacemi nad daty.

4 Návrh databáze

Na následujících řádcích bude popsáno databázové schéma, entity, které se v databázi nacházejí a relace, které jsou pro jednotlivé entity definovány.

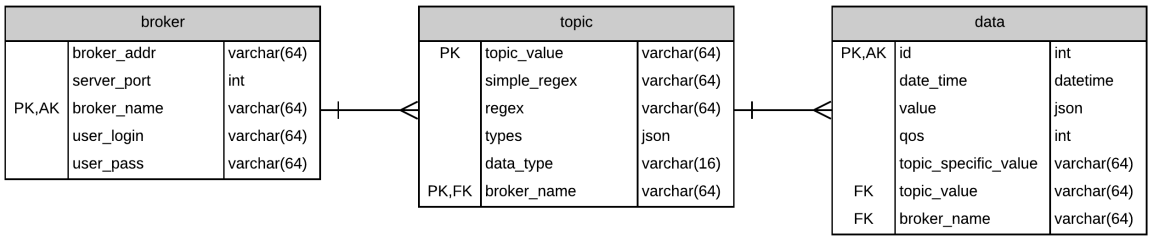
4.1 Entity v databázi

Databáze obsahuje tři entity: broker, topic a data.

Entita broker, reprezentuje brokery, ke kterým je program připojen, a odebírá z nich zprávy. Z toho vyplývá, že bude možno odebírat data z více různých brokerů současně. Entita obsahuje informace o adrese a portu brokeru, přihlašovací údaje a také název, pod kterým bude daný broker v programu identifikován. Název brokeru v rámci programu je také primárním klíčem, musí být tedy jedinečný. Entita broker je ve vztahu s další entitou zvanou topic (téma).

Jedná se o 1:N kardinalitu, tedy broker může mít N témat. Z toho vyplývá, že entita topic obsahuje cizí klíč referující na název brokeru. Mimo tento cizí klíč obsahuje entita informace o odebíraných tématech, tedy identifikátor tématu, regulární výrazy, na základě kterých budou data archivována a obsahuje rovněž informaci o datových typech archivovaných dat. Identifikátor odebíraného tématu a cizí klíč obsahující název brokeru tvoří složený primární klíč entity. Nepředpokládá se totiž, že by nastala situace, kdy by docházelo k odběru stejného tématu v rámci jednoho brokeru opakovaně.

Entitu topic spojuje 1:N kardinalita s entitou data, která představuje datovou jednotku, určenou k archivaci. Z jednoho topicu tedy může být přijato N datových jednotek. Primárním klíčem entity data je ID, které je představováno automaticky inkrementovaným celým číslem. Entita data má dva cizí klíče - prvním je název brokeru, ze kterého archivovaná informace pochází, druhým cízím klíčem je odkaz na entitu topic, konkrétně tedy na identifikátor odebíraného tématu. Entita rovněž obsahuje atribut identifikující konkrétní topic, ze kterého zpráva přišla, jelikož již zmíněný odkaz na entitu topic může obsahovat některý z wildcard znaků, a nebylo by tedy možné určit, z jakého tématu daná zpráva pochází. Tato informace je vyextrahována přímo z MQTT datagramu. Další informace, které se archivují a jsou rovněž vyjmuty z přijaté MQTT zprávy, jsou hodnota zprávy a QoS. Entita topic obsahuje také datum vytvoření záznamu. Entity jsou vizualizovány v diagramu entit na obrázku 3.



Obrázek 3: Diagram entit

5 Návrh programu

Tato část textu bude zaměřena na praktický návrh a implementaci programu pro archivaci dat přenášených přes MQTT Broker.

5.1 Výběr dat k archivaci

Jedním z prvních problémů, které bylo nutno vyřešit, bylo navrhnout způsob, na základě kterého bude určováno, jaká data budou archivována. Uvažujme tedy zprávu, jejíž datová část (payload) bude obsahovat následující data `"temperature: 90"`. Prvotní myšlenkou bylo, ukládat do databáze surová data, to znamená, že by se příchozí data nijak nezpracovávala. Pokud by nad daty byly prováděny operace, například nalezení nejmenší teploty z určitého rozsahu, musel by být každý záznam zpracováván samostatně, uložen do vhodné datové struktury a nad ní následně proveden výpočet. To by bylo zdlouhavé.

Ideálním řešením by bylo, mít v tabulce numerická data, nad kterými by bylo vhodné provádět tyto operace přímo na úrovni databázového systému. Toho by bylo možné docílit, pokud by data byla zpracovávána přímo během procesu archivace. Do databáze by se tedy dostaly pouze numerické hodnoty.

Bylo proto nutné vymyslet systém, na základě kterého uživatel definuje část příchozích dat, kterou potřebuje archivovat. Je potřeba brát v úvahu, že formát přijatých dat může být různý. Vhodným řešením se zdálo být nadefinování regulárního výrazu, kterým uživatel určí jak formát zprávy, tak část zprávy, kterou chce archivovat. Pokud tedy zpráva nesplňuje regulární výraz, je ignorována, pokud zpráva regulární výraz splňuje, je z ní vyextrahována požadovaná část a ta je následně uložena do databáze. Pro zlepšení uživatelské zkušenosti byl zaveden takzvaný "zjednodušený regulární výraz", ten je inspirován funkcí `printf()` z programovacího jazyka C, kde programátor definuje pomocí značek, jakého typu jsou proměnné, které se chystá do výpisu dosadit, a kde se budou nacházet. Zjednodušený výraz je pak pro potřeby aplikace přeložen na běžný regulární výraz.

5.2 Datové typy archivovaných dat

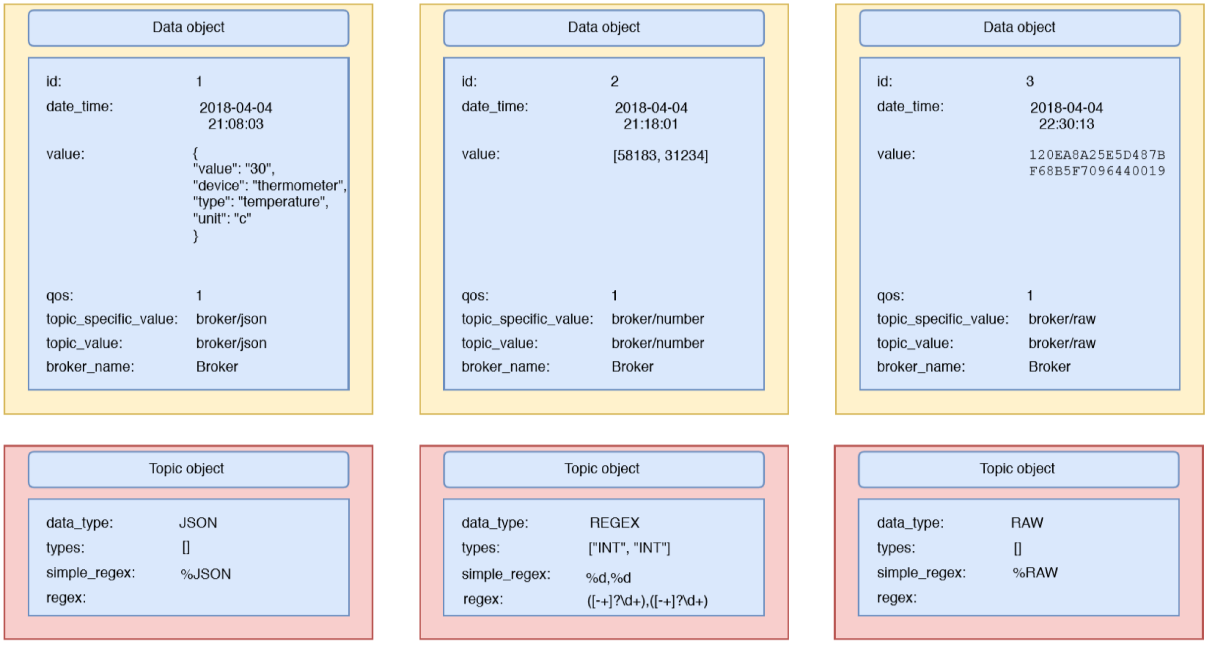
V produkčním nasazení ale může být požadován i sběr jiných než číselných dat. Bylo tedy nutno vymyslet, jak archivovat data tak, aby statistické operace nad číselnými údaji, mohly být prováděny na databázové úrovni, a zároveň archivovat i nenumerická data.

MySQL od verze 5.7 disponuje nativní podporou JSONu jako datového typu. Takže je možné do sloupce tabulky uložit celý JSON dokument a nad ním poté provádět běžné databázové operace. MySQL při vkládání takových dat automaticky validuje, jestli se jedná o korektní JSON zápis. Uložené JSON dokumenty jsou převedeny na interní binární formát, který umožňuje rychlý přístup k prvkům dokumentů. Data tedy nemusí být parsována z textové reprezentace. Binární

formát je strukturován tak, že umožní serveru procházet subobjekty nebo vnořené hodnoty pomocí klíče nebo indexu pole, nemusí tedy načítat všechny hodnoty JSON dokumentu [11].

Tato nová funkcionalita databázového systému MySQL byla velice příhodná. Systém na základě zadaného datového typu uživatele ví, jaké datové typy se v daných tématech budou nacházet, a tak k nim bude i přistupovat. Data jsou vždy uložena v JSONu, do kterého je možné uložit data všech datových typů. Příklad takových dat je možno vidět na obrázku 5, kde jsou vizualizovány objekty uložené v databázi, aby bylo možno vidět, jaké informace jsou uloženy v atributu `value`.

Průběh zpracování příchozí MQTT zprávy a její následná archivace je znázorněna na obrázku 4.

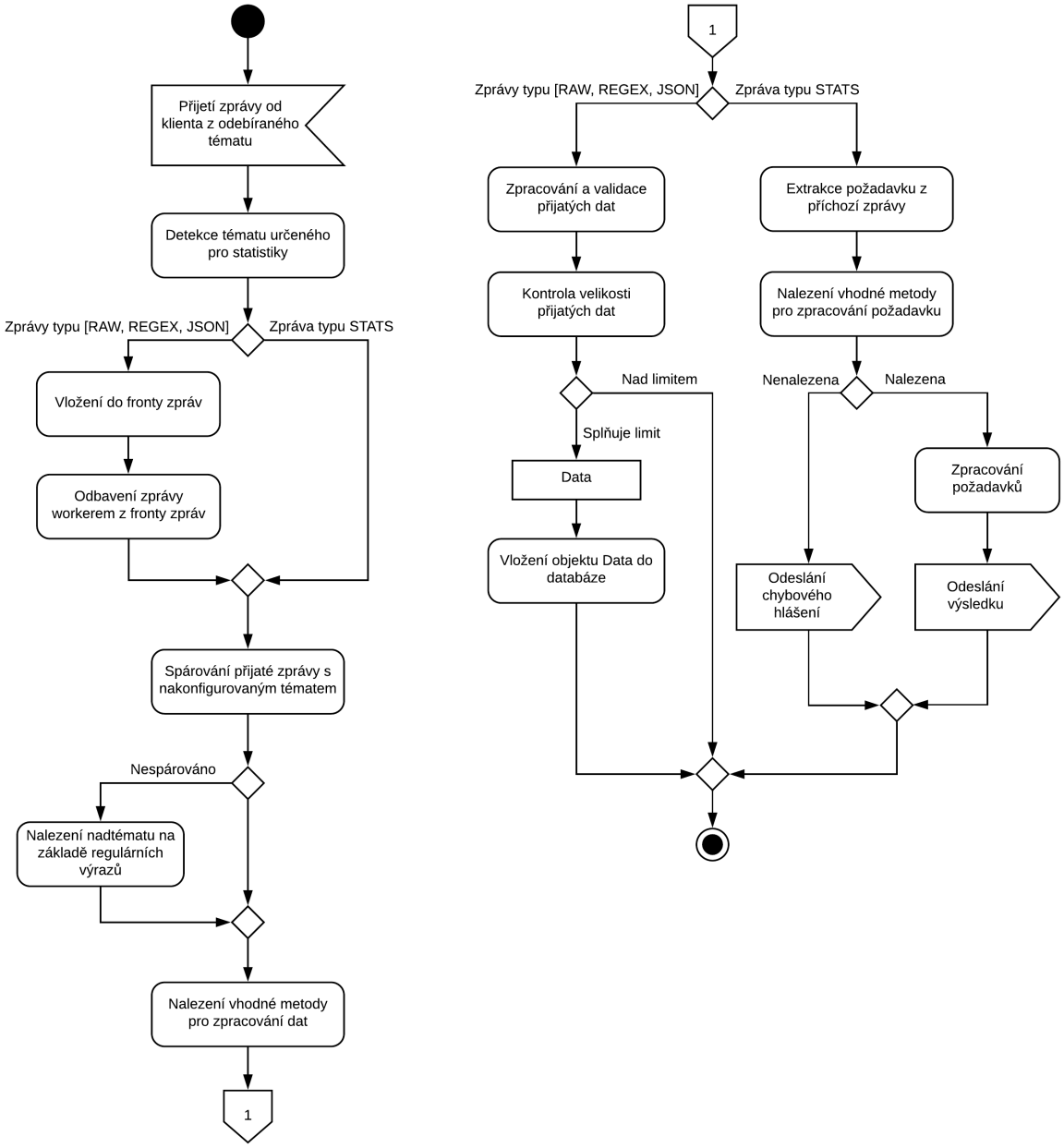


Obrázek 5: Ukázka datových objektů uložených v databázi, včetně souvisejících objektů typu topic

5.3 Rozhraní pro přístup k archivovaným datům

Program tedy disponoval fukcemi pro archivaci dat několika typů. Dále bylo nutné vymyslet způsob, jakým bude uživatel přistupovat k archivovaným datům. Vzhledem k zaměření této práce padla volba opět na protokol MQTT.

Idea byla taková, že aplikace na základě přijaté zprávy (dotazu) vrátí odesílateli odpověď. Byl zaveden nový typ zpráv - řídící zprávy, které se nebudou archivovat, ale budou zpracovány a poté na ně aplikace odpoví. Rovněž byla přidána položka ke konfiguraci, kde uživatel definuje dotazovací téma, tedy téma, kam se budou dotazy (řídící zprávy) posílat; na základě toho aplikace disponuje informací o tom, kam budou dotazy chodit.



Obrázek 4: Activity diagram znázorňující archivaci příchozí zprávy

V okamžiku, kdy dotaz příjde do uživatelem specifikovaného tématu, je řetězec zpracován a uživateli jsou vracena data v JSON zápisu. Data jsou vrácena do nadtématu, sloužícího pro odpověď. Pro korektní přijetí odpovědi je tedy nutno odebírat toto téma.

Jelikož není zcela běžným řešením přistupovat k datům pomocí MQTT protokolu, bylo rovněž implementováno REST API, které nabízí konzervativnější přístup k datům.

5.4 Grafická vizualizace dat

Jedním z požadavků této práce bylo vhodné grafické vizualizování archivovaných dat. Od programu běžícího na pozadí a archivujícího data se ale taková funkcionalita neočekává. Byl tedy implementován klientský program, který je součástí řešení, ten umožňuje grafy na základě dat ze serveru (program pro sběr dat) generovat. Server vrací v JSON zápisu jednotlivé osy grafu, ty jsou poté vykresleny.

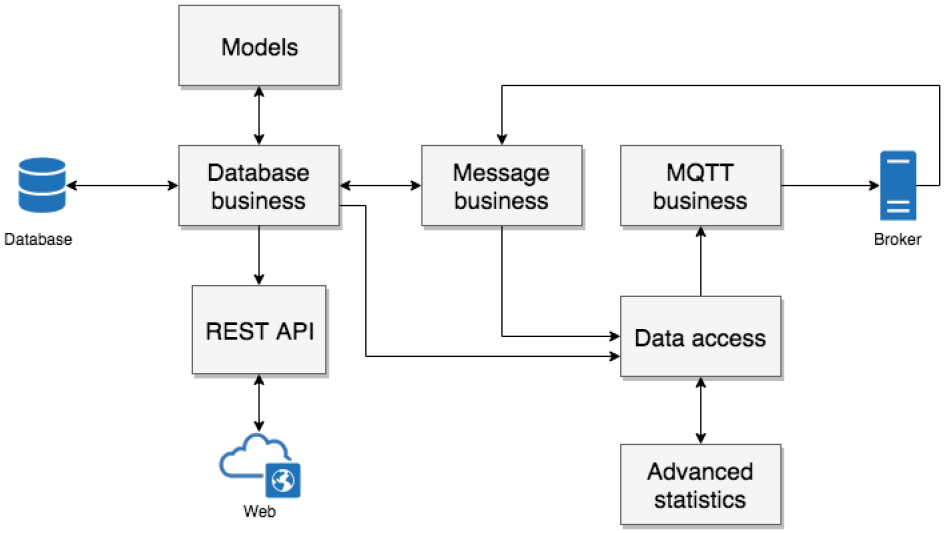
5.5 Zátěžové testy

V okamžiku, kdy byl program pro archivaci dat vystaven zátěžovým testům, začalo při velmi vysoké frekvenci příchodu MQTT zpráv docházet k jejich zahazování. To bylo řešeno vytvořením fronty zpráv, která je odbavována několika procesy.

Cílem této kapitoly bylo obecné popsání vývojového procesu celého systému. Byly zde položeny otázky, které bylo nutno zodpovědět, aby bylo možné přejít k implementaci systému. Technické detaily jsou popsány v následující kapitole.

6 Moduly systému

Kvůli systematičtějšímu vývoji a lepšímu přehledu nad vyvíjeným programem je program rozdělen na několik modulů, což je možné vidět na obrázku 6, kde šipky vyjadřují tok dat mezi jednotlivými moduly.



Obrázek 6: Přehled komponent

6.1 Modul Database business

Databázová vrstva obstarává potřebnou logiku, kterou je potřeba provádět pro korektní integraci aplikační logiky a databáze. Vzhledem k použitému ORM modulu SQLAlchemy se mapování tabulek na objekty provádí automaticky.

Modul Database business obsahuje třídu pro vytváření plánovaných úloh (sheduled tasks), třídu pro mazání dat na základě specifických kritérií a, v neposlední řadě, také třídu pro přístup k nasbíraným datům.

Při inicializaci modulu Database business jsou načteny potřebné informace z konfiguračního souboru. Soubor je uložen ve složce s konfiguračními soubory `config` a nazývá se `database.ini`. Mimo základních údajů, které jsou nutné pro připojení k SŘBD, jsou v souboru také položky reprezentujíci některé další možnosti, mezi něž patří například typ SQL databáze. Použití SQLAlchemy totiž umožnuje použití různých SŘBD. Mezi další možnosti patří nastavení maximální velikosti přijaté zprávy v bajtech a maximální stáří archivovaných zpráv ve dnech.

Po načtení konfiguračního souboru dojde k inicializaci databáze na základě modelů (models). Modely představují objekty pro objektově relační mapování a jsou jakýmisi držiteli informací. Pokud inicializace probíhá nad prázdnou databází, jsou v ní automaticky vytvořeny potřebné tabulky a relace.

Následně je vytvořena plánovaná úloha. Tato konkrétní plánovaná úloha má za úkol každou hodinu vymazat veškerá data z databáze starší než předem definovaný počet dnů. Počet dnů je zadán v databázovém konfiguračním souboru. Plánovaná úloha se přepisuje pouze při spuštění programu. To znamená, že změna podmínky plánované úlohy vyžaduje restart programu.

Moduly, které chtějí do databáze uložit data, je musí modulu Database business doručit ve formě instance modelů. Modul Database business pak tyto instance v rámci databázové relace uloží do databáze.

Pro přístup k archivovaným datům se využívá SQLAlchemy Query API. To se dá společně s metodami pro filtrování používat obdobným způsobem jako selekce dat v jazyce SQL. Výsledná data jsou vrácena v podobě instancí modelů. Pokud je při dotazování použita formule *join*, obsahují výsledné instance zároveň i instance modelů sdílejících stejnou relaci. Pokud *join* není použit, obsahuje instance pouze cízí klíče.

6.2 Modul Message business

V následující kapitole bude představen modul Message business, ten se zabývá zpracováním zpráv přenýšených pomocí MQTT protokolu.

6.2.1 Obecný popis modulu

Při návrhu programu bylo nutno důkladně promyslet, jakým způsobem definovat témata, která chce uživatel odebírat a následně archivovat. Rovněž bylo nutno nějakým způsobem definovat, jaký typ dat se v daném tématu vyskytuje, případně jak odebírat jenom určitý typ dat a ignorovat data, která podmínku nesplňují, ale vyskytují se v tomtéž tématu. Takovéto datové anomálie by mohly způsobit určitý druh nekonzistence dat v databázi.

Aby byl program schopen archivovat pouze chtěná data, je nutné, aby uživatel znal formát příchozích zpráv a na základě toho správně zvolil některý z typů zpráv. Odebíraná témata jsou definována v konfiguračním souboru, který se nachází v adresáři `config`, který lze najít v adresáři s programem. Soubor se nazývá `brokers.ini` a uživatelé v něm definují informace o brokeru, ze kterého budou následně odebírána data. Rovněž je v něm definováno pole odebíraných témat. Odebírané téma se skládá z identifikátoru daného tématu a ze zjednodušeného regulárního výrazu, který definuje formát zpracovávaných dat.

Zjednodušený regulární výraz zamezuje archivaci dat, která nesplňují požadovaný formát. Zjednodušený regulární výraz zároveň slouží jako předpis, který bude zohledněn při budoucím zpracování dat. Uživatel může zvolit z následujících typů zjednodušených regulárních výrazů pro odebírané téma: JSON, RAW a nebo REGEX.

JSON Pokud je zpráva přijatá z tématu odchytávána zjednodušeným regulárním výrazem typu JSON, prochází přijatá zpráva validací. Validace zprávy v tomto konkrétním případě zna-

mená, že je zjišťováno, jestli je obsahem přijaté zprávy validní JSON. Pokud se o validní JSON nejedná, není zpráva archivována.

RAW V případě, že je zpráva přijatá z tématu odchytávána zjednodušeným regulárním výrazem typu RAW, znamená to, že přijatá data nebudou nijak kontrolována a budou rovnou archivována. Tento typ zjednodušeného výrazu je vhodný například pro archivaci obrázků nebo případně pro data, která nebudou dále nějak statisticky zpracovávána.

REGEX Pokud probíhá odběr dat, která splňují určitý vzor, je možné z těchto dat vyselektovat jenom ty části, které máme zájem archivovat. To je možné udělat pomocí zjednodušeného regulárního výrazu typu REGEX.

Zjednodušený regulární výraz typu REGEX je narozdíl od předchozích zjednodušených regulárních výrazů, zpracováván jiným způsobem. Uživatel za pomocí zástupných znaků definuje části řetězce, které chce archivovat. Tyto zástupné znaky vypadají takto: %d (decimal), %f (float), %b (boolean) a %s (string). Je žádoucí zvolit korektní datový typ, jelikož od něj se budou dále odvíjet například statistické či jiné početní operace.

Mějme například příchozí data v následujícím formátu:

```
temperature_in:25, temperature_out:3
```

Naší snahou je extrahovat číselné údaje o obou teplotách a poté například vykreslit graf, který bude vizualizovat změny teplot v závislosti na čase. Aby bylo možné číselné hodnoty korektně načíst, je nutno využít zjednodušeného regulárního výrazu typu REGEX. Z ukázky je možno vidět, že příchozí data obsahují zaokrouhlená celá čísla, je tedy žádoucí zvolit značku decimal (%d).

Výsledné nastavení odběru by tedy vypadalo následovně:

```
topics = ["regex/#;temperature_in:%d, temperature_out:%d"]
```

Data, která ve výsledku budou archivována a bude je možné použít pro statistické operace, případně vizualizace pomocí grafů, budou vypadat takto:

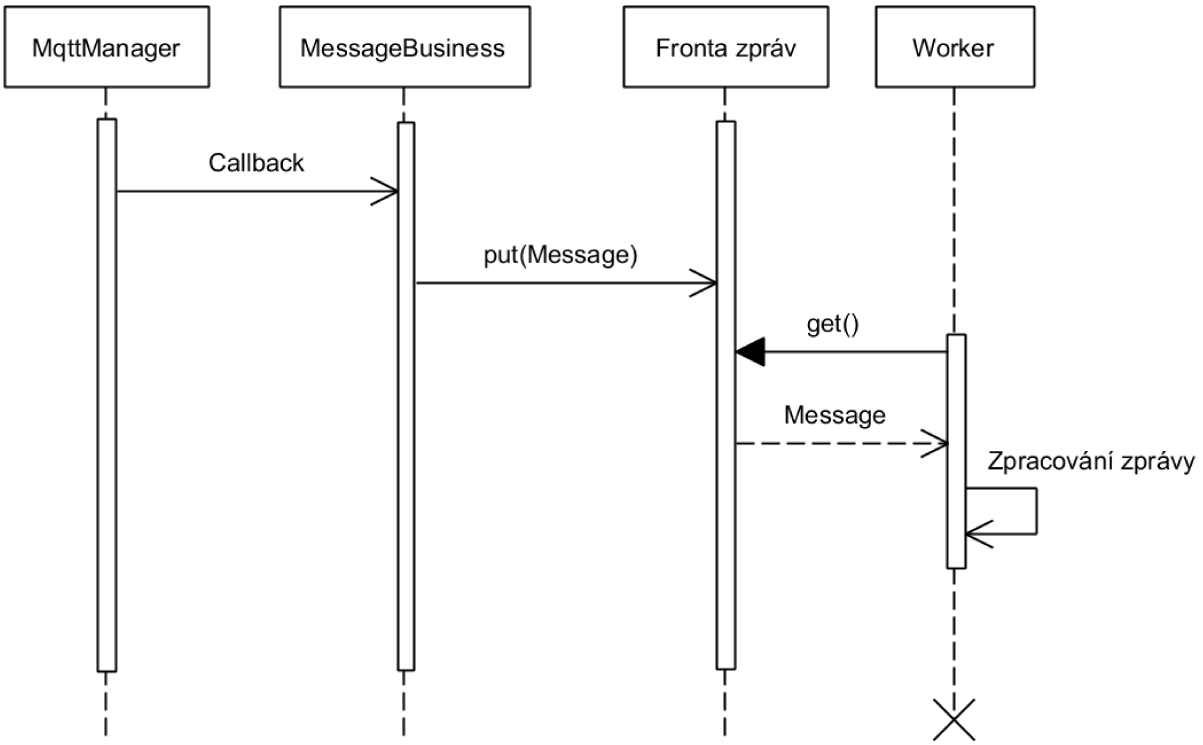
```
[25,3]
```

6.2.2 Funkční popis modulu

Modul tedy zpracovává přijaté zprávy z odebíraných témat a poté za pomocí modulu Database business zajišťuje uložení dat do databáze. Modul je rovněž zodpovědný za evaluaci zjednodušených regulárných výrazů.

V okamžiku, kdy je zpráva předána modulu Message business, je modulem uložena do objektu Queue, který představuje frontu zpráv. Queue je součástí nativního Python modulu multiprocessing. Tento modul slouží pro vytváření paralelních procesů na úrovni operačního systému a

konkrétně objekt Queue je thread-safe implementace fronty zpráv v operačním systému. Fronta zpráv má předem nastavenou velikost. Velikost fronty a množství procesů, které budou frontu obsluhovat, dále jen worker pool, lze specifikovat v konfiguračním souboru. Velikost fronty je teoreticky neomezená, stejně tak množství procesů, obsluhujících frontu. Nastavení by mělo být zvoleno s ohledem na hardwarové vybavení počítače. Průběh zpracování přijatých MQTT zpráv je vizualizován formou sekvenčního diagram na obrázku 7, vizualizace fronty zpráv a worker poolu je potom na obrázku 8.

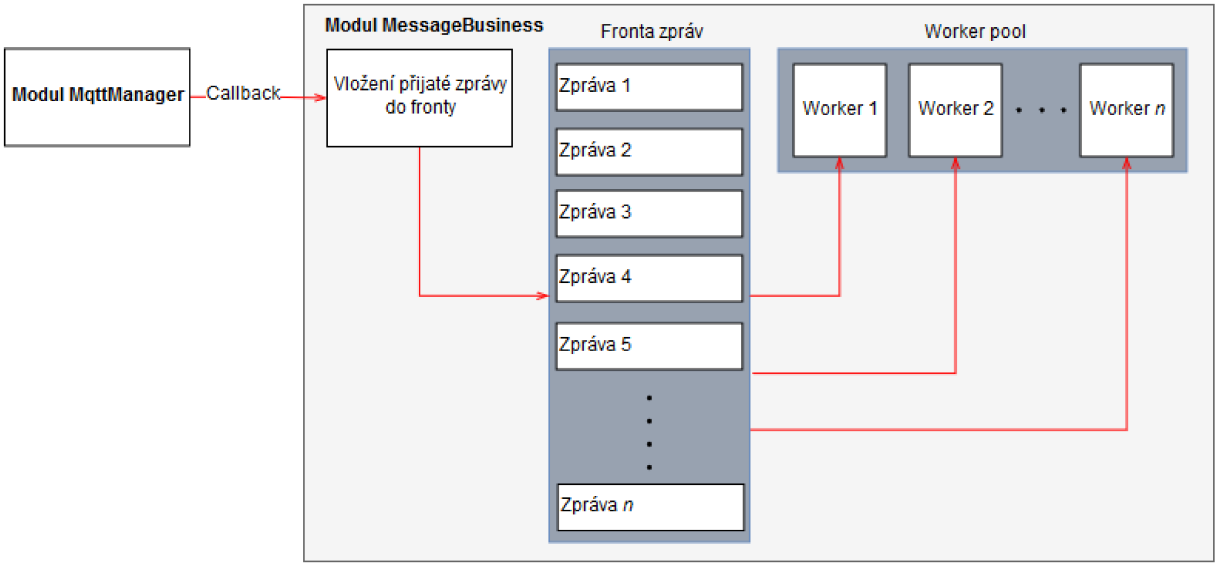


Obrázek 7: Průběh zpracovávání přijatých MQTT zpráv.

Následně je zpráva odebrána z fronty jedním z procesů. Na základě tématu, ze kterého zpráva přišla, je zavolána metoda pro specifický typ zprávy, tedy v závislosti na zvoleném zjednodušeném výrazu pro dané téma.

Je zde tedy implementace metody pro zpracování zpráv typu RAW. V tomto případě se v metodě zkontroluje, zda přijatá zpráva nepřesahuje maximální velikost zprávy a pokud je podmínka splněna, dojde ke konstrukci datového objektu (Data). Konstrukce objektu se provádí na základě dat z přijaté zprávy a objektu typu Topic. Jelikož se jedná o surová data, neprovádí se žádné další operace nad přijatou zprávou a dojde pouze k zapouzdření.

Situaci, kdy z odebíraného tématu příjde zpráva typu JSON, řeší metoda, která je dedikována pro tento typ zpráv. Opět dojde ke kontrole velikosti zprávy a za předpokladu, že podmínka není splněna, je zpráva zahozena. V opačném případě dojde k vybalení informací z obdržené zprávy a je zkonstruován objekt typu Data, atribut value (hodnota) objektu data, ale obsahuje přijaté



Obrázek 8: Vizualizace fronty zpráv a worker poolu.

informace deserializované do formátu typu JSON. Pokud deserializace selže a přijatá data nejsou data reprezentována JSON zápisem, jsou data zahozena. Archivace nevalidních dat by způsobila nemožnost provádět statistické a další operace nad archivovanými daty. Žádoucím výsledkem je tedy záznam v datové tabulce, který v atributu value (hodnota) obsahuje JSON, jak je možno vidět na obrázku 9.

```
object
   broker_name : TestingBroker
   date_time : 2018-03-25 11:42:33
   id   : 2954
   qos  : 0
   topic_specific_value : iot-devices/living-room/heat-sensor
   topic_value : iot-devices/living-room/#
  "value": {
    "device": "heat_sensor",
    "temperature": 39,
    "uptime":"101 days"
  }
```

Obrázek 9: Ukázka objektu typu data, obsahující JSON v atributu value (hodnota)

Posledním typem zprávy může být zpráva typu REGEX. Pro rekapitulaci je dobré uvést, že při definici témat k odběru a použití zjednodušeného regulárního výrazu typu REGEX, je nutné zástupnými znaky označit ty části řetězce, které budou archivovány. To znamená, že znalost formátů přijatých zpráv je nutností. V následujícím pododstavci bude vysvětlen proces archivace

zpráv typu REGEX.

Jak již bylo naznačeno, zpracování zprávy typu REGEX není tak triviální jako zpracování zprávy typu RAW nebo JSON. Rozličným způsobem je vytvářen objekt Topic. Ten obsahuje atribut obsahující zjednodušený regulární výraz, regulární výraz a datové typy. Při vytváření objektu Topic (téma) je načten zjednodušený regulární výraz, ten je přeložen na běžný regulární výraz, na základě kterého dochází ke zpracovávání přijaté zprávy.

V okamžiku, kdy je zpráva přijata a aplikace detekuje, že je z tématu odebírající zprávy typu REGEX, dojde k evaluaci zprávy na základě nezjednodušeného regulárního výrazu. Pokud data nekorespondují s nezjednodušeným regulárním výrazem, jsou zahozena. V opačném případě jsou požadované informace pomocí nezjednodušeného regulárního výrazu ze zprávy vyextrahovány a uloženy v databázi.

6.3 Modul MQTT business

Modul MQTT business je postaven na klientské knihovně Paho MQTT a obsahuje implementaci metod pro odesílání a odběr zpráv. Pokud přijde zpráva do tématu, na který je nastavený odběr, zavolá se callback. Zpráva je pak následně zpracovávána dalšími moduly.

6.4 REST API

Volba modulu SQLAlchemy a modulu Flask s sebou přinesla výhodu ve formě velice jednoduché implementace REST API. Uživatel je tedy schopen přištupovat k archivovaným datům pomocí REST API. Data je možné získat na základě názvu brokeru, konkrétního tématu a poté je možné data získat také specifikováním rozsahu. Rozsah dat se dá specifikovat buďto rozsahem, nebo je možné získat data za posledních několik let, měsíců, dní, hodin, minut nebo vteřin. HTTP dotaz pak může vypadat takto:

```
http://127.0.0.1/<BROKER_NAME><TOPIC_NAME>?last=5s
```

BROKER_NAME představuje název brokeru v rámci aplikace a TOPIC_NAME představuje řetězec identifikující konkrétní odebírané téma. Dotaz vrátí data, která byla nasbírána za posledních 5 vteřin. Pokud databáze tato data z nějakého důvodu neobsahuje, je uživateli vrácena zpráva informující o této skutečnosti.

Jak již bylo zmíněno, množství obdržených dat lze ovlivnit rovněž specifikací datového rozsahu:

```
http://127.0.0.1/testingbroker/esp_hub/device/4b131a/telemetry?
from=2018-02-05-18:55:03&to=2018-02-05-20:00:00
```

Pokud je cílem získat veškerá data z daného tématu a brokeru, je možné rovněž použít REST API. Stačí pouze vynechat dotaz v URL, tedy:

```
http://127.0.0.1/testingbroker/esp_hub/device/4b131a/telemetry
```

6.5 Modul Data access

Data access modul slouží k přístupu k nasbíraným datům. Data jsou získána pomocí MQTT protokolu.

Uživatel nadefinuje v konfiguračním souboru pro brokery téma, do kterého budou odesílány dotazy. Odpověď ve formě dat odpovídající danému dotazu příjde o úroveň výše do nadtématu response (odpověď).

```
# odpověď příjde do tématu testingbroker/statistics/response
statistics = "testingbroker/statistics"
```

Pro tento účel byl vytvořen nový typ zpráv, takže mimo zprávy typu JSON, RAW a REGEX je zde ještě speciální typ zpráv STATS. Ten je použít automaticky pro všechny zprávy, které budou poslány na specifikované téma určené pro statistiky.

Zpracování informace o statistickém tématu z konfiguračního souboru probíhá stejným způsobem jako definice témat, ze kterých mají přicházet zprávy určené k archivaci. Odpadá ale definice datového typu pomocí zjednodušeného regulárního výrazu. Ten je nastaven implicitně na STATS.

V okamžiku kdy přijde zpráva typu STATS do modulu Message business, zavolá se metoda určená ke zpracování tohoto typu zpráv. Zpráva je poté rozparsována a na základě získaných informací jsou výsledky vráceny do nadtématu response.

Pokud by uživatel požadoval, aby byl dotaz nějakým způsobem identifikován a aby odpověď byla určena přímo jeho požadavku, je možné použít například identifikátor.

Použití identifikátoru je dobrovolné. Pokud ale použit nebude, všechny odpovědi budou posílány do stejného tématu, což by bylo v případě, kdy program používá více uživatelů současně, nevhodné.

6.5.1 Dotazovací jazyk

Aby měly dotazy formu a byly jednoduše zpracovatelné, byl vytvořen jednoduchý dotazovací jazyk pro potřeby dotazování se v rámci této aplikace.

Pomocí dotazovacího jazyka může uživatel získat data na základě názvu brokeru, tématu a specifikace datového rozsahu. Dotazovací jazyk podporuje také základní operace, které jsou známé například z SQL, tedy najít maximum nebo minimum a vypočítat průměr nebo sumu. Pokud je v databázi uložen například JSON, lze tyto operace provádět také, a to specifikací klíče, na základě kterého se určí ten JSON atribut, nad kterým se má daná operace provést.

get-help Tento příkaz vrátí řetězec obsahující nápovědu, která obsahuje informace o dostupných příkazech a syntaxích dotazů.

get-stats Tento příkaz vrací data, případně výsledky některých základních operací, prostřednictvím MQTT. Aby bylo možné tento příkaz používat, je nutno jej doplnit o parametry. Povinným parametrem je název brokeru v rámci aplikace a identifikace tématu, ze kterého jsou data požadována. Toto použití vrátí veškará data ze specifikovaného tématu a brokeru.

Pokud je potřeba získat data za určité období je nutno použít parametry `from` a `to`. Pokud bude vynechán parametr `to`, bude implicitně nahrazen současným časem a datem. Dalším dobrovolným parametrem, který lze použít, je parametr `last`, díky kterému je možné získat data archivovaná za posledních n let, měsíců, týdnů, dnů, hodin, minut nebo vteřin.

Rovněž je možné nad danými daty provádět některou z jednoduchých operací, například získat minimum, maximum, průměr nebo sumu. Pro tyto účely slouží parametr `ops`. V případě, že jsou data uložená v JSON, je možné parametrem `key` definovat atribut, ze kterého mají být data načtena. Operace fungují rovněž nad daty, ktera jsou uložena v poli.

```
# Výpočet průměru nad daty, ktera jsou uložena v JSON za posledních sedm dní
get-stats testingbroker esp_hub/device/950316592/data ops avg last 7d key value
```

get-graph Program také umožňuje generování grafů. Ke generování grafů slouží modul Matplotlib. Data, na základě kterých je graf sestrojen, jsou získána stejným způsobem jako v předešlém případě; množina získaných dat je poté předána modulu zodpovědnému za generování grafů, graf je následně vygenerován.

Problémem je, že není možné generovaný graf zobrazit v příkazové řádce nebo terminálu. Je zde tedy možnost použít parametr `send_as_bits`, kdy je graf po vytvoření převeden na bity a následně odeslán pomocí MQTT. Uživatel může takto zrekonstruovat graf a použít jej ve své aplikaci. Lze také použit parametr `save_plot`, kdy dojde k uložení grafu jako obrázkového souboru na zařízení, kde probíhá sběr dat. Pokud graf není uložen do souboru, nebo odeslán jako bitová reprezentace, je přes MQTT odeslána odpověď obsahující honoty osy x a osy y.

Jelikož se nejedná o pohodlný způsob získávání grafů z nasbíraných dat, je součástí systému také klientská aplikace, která data vizualizuje. Té je věnována samostatná kapitola.

Příkaz pro získání grafů má také parametr specifikující typ křivky generovaných grafů. Rovněž je možné specifikovat text, kterým budou označeny osy grafu.

get-overview Příkaz vracející výpis počtu archivovaných zpráv pro jednotlivá témata ve všech brokerech. Obsahuje rovněž počet všech zpráv nasbíraných v rámci brokeru a také počet zpráv nasbíraných všemi brokery. Příkaz slouží k získání obecného přehledu o archivovaných datech, případně pro ladění nastavení odebíraných témat.

delete-from-db Tento příkaz slouží k mazání archivovaných dat nebo témat. Používá se v kombinaci s povinnými parametry obsahujícími název brokeru a téma. Pokud budou použity pouze tyto dva povinné parametry, dojde ke smazání celého tématu včetně dat. V případě, že

budou použity dobrovolné parametry `from` a `to`, budou smazaná jenom data odpovídající datovému rozsahu. Pokud uživatel vynechá parametr `to`, budou smazána data z intervalu začínající datovým údajem z parametru `from` až do současné chvíle. Časový údaj může být rovněž specifikován parametrem `last`, kdy budou smazána data archivovaná za posledních n let, měsíců, týdnů, dnů, hodin, minut nebo vteřin.

6.6 Modul Advanced statistics

Modul, který obsahuje implementace metod pro lineární regresi a predikci.

6.6.1 Lineární regrese

Modulu jsou předána data, na základě kterých je vygenerován dvourozměrný graf. Modul používá balíček pro strojové učení scikit-learn a balíček NumPy, ze kterého je použita datová struktura, vhodná pro takovéto výpočty.

Pro vygenerování takového grafu je využita metoda nejmenších čtverců. Cílem tedy je nakreslit přímku, která bude co nejlépe minimalizovat reziduální sumu čtverců mezi použitými daty a odezvami předpovězené lineární aproximací. Rovněž jsou vypočteny koeficienty, reziduální součet čtverců a rozptylové skóre.

Metoda generující grafy lineární regrese přijímá slovník obsahující pole hodnot a datumů, datový typ (REGEX nebo JSON) a případně také klíč, pokud se jedná o data zapsána v JSON formátu.

Datumy jsou následně převedeny na čísla, aby je bylo možné použít pro matematické modely. Datumy jsou poté vloženy do jednorozměrného pole modulu NumPy. Jedná se o hodnoty pro osu x. Takto upravené datumy jsou poté společně s hodnotami použity pro volání metody `train_test_split`, která je obsažena v modulu scikit-learn. Tato metoda rozdělí pole na menší pole pro trénování a testování.

Následně je vytvořen objekt `LinearRegression` knihovny scikit-learn, objekt je naplněn menšími poli, která byla vytvořena v předchozím kroku, konkrétně těmi, která obsahují data vhodná pro trénink. Na základě toho získáme data pro sestrojení grafu lineární regrese.

Jelikož je přístup k nasbíraným datům zprostředkován pomoci MQTT příkazů, je problém graf vizualizovat. Modul vrací několik polí, představujících jednotlivé osy grafů. Uživatel tedy bude muset přijatá data zpracovat a graf zkonstruovat. Konstrukce grafu lineární regrese je rovněž jednou z funkcionalit klientské aplikace, které se věnuje samostatná kapitola.

6.6.2 Predikce

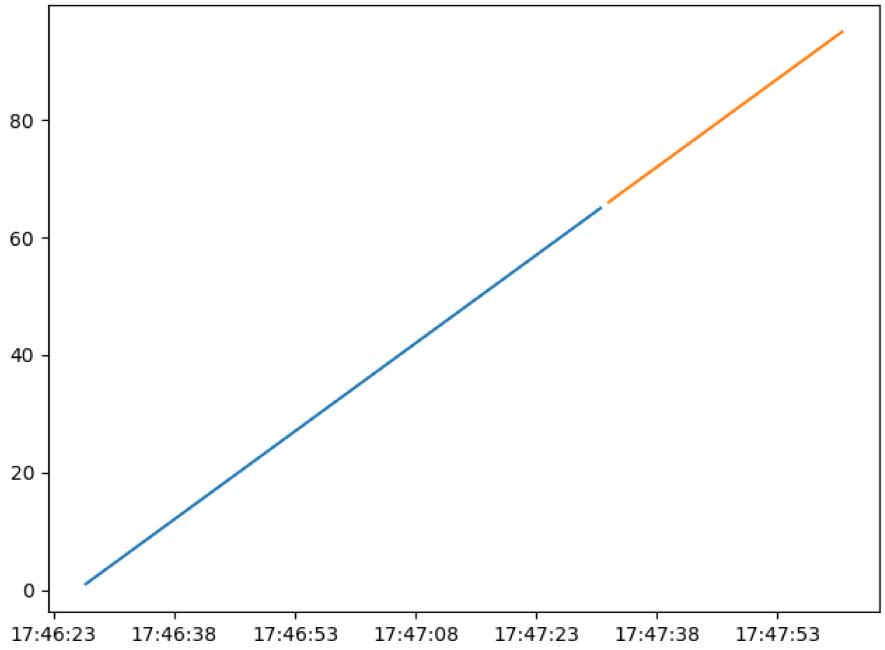
V systému je také k dispozici možnost predikovat data. Uživatel zadá interval, v jakém byla data archivována, dále zadá množství dat, která budou predikována, výstupem budou data zapsána ve formátu JSON, obsahující datumy, predikované datumy, archivovaná data, predikovaná data a koeficient spolehlivosti, který udává, jak moc je predikce akurátní.

Metoda provádějící predikci přijímá slovník obsahující pole hodnot a datumů, datový typ (REGEX nebo JSON) a případně také klíč, pro data zapsána v JSON formátu. Pro predikci jsou použity moduly Pandas, scikit-learn

Data jsou zprvu rozdělená na datumy a hodnoty. Z nich je poté vytvořený objekt `DataFrame` modulu Pandas, kde datumy jsou indexem datové struktury.

Data jsou v rámci přípravného procesu optimalizována, jsou tedy například odstraněna odlehlá pozorování a podobně. Následně jsou data rozdělena na podmnožiny pro testování a trénink. V dalším kroce je vytvořen objekt `LinearRegression`, ten je naplněn podmnožinami pro trénink. Jedná se o objekt knihovny scikit-learn pro strojové učení a funguje stejně jako v případě implementace generování grafů lineární regrese v tomto modulu. Množiny pro testování jsou následně použity pro výpočet koeficientu spolehlivosti. Nakonec je zavolána metoda `predict` objektu `LinearRegression`, která vrátí predikované hodnoty.

Na obrázku 10 lze vidět ukázkový graf, který byl vygenerován na základě výsledků predikce, implementované v rámci systému. Použitá data byla číselná řada začínající jedničkou a končící číslem sedmdesát, mezi jednotlivými hodnotami jsou vteřinové intervaly. Požadována byla predikce hodnot v následujících třiceti sekundách, výsledná predikce má koeficient spolehlivosti roven jedné, jelikož se jedná o data s lineárním růstem.



Obrázek 10: Graf sestrojen z původních dat (modrá osa) a predikovaných dat (oranžová osa)

7 Klientský program

Jak již bylo několikrát zmíněno, je problémem graficky vizualizovat data, která jsou získána pomocí dotazovacího jazyka, vytvořeného pro potřeby tohoto projektu. To bylo motivací pro vytvoření klientského programu, který na základě dotazů zobrazuje interaktivní grafy. Další výhodou, kterou klientská aplikace uživateli přináší, je snadnější psaní dotazů. V této kapitole bude přiblížen klientský program a jeho implementace.

7.1 Popis programu

Klientský program slouží jako jakési rozhraní pro přístup k datům. Uživatelův dotaz je přeložen do syntaxe dotazovacího jazyka a následně odeslán na broker. Odeslán je do tématu označeného unikátním identifikátorem, a to proto, aby k datům mohlo pomocí aplikace přistupovat více uživatelů současně. Odpověď přijde do nadtématu s konkrétním identifikátorem, odpověď tedy bude, stejně jako dotaz, izolována. Pokud odpověď nedorazí ve specifikovaném intervalu, už se na ní dále nečeká a je nutné dotaz zopakovat. Program tedy představuje klienta z architektury klient-server, server je zase představován programem pro sběr dat, který klientskému programu poskytuje data k vizualizaci.

Jelikož je dotazovací jazyk na úkor jeho minimalističnosti relativně neintuitivní, bylo snahou udělat jej pomocí klientské aplikace trochu přístupnější uživateli. Program tedy disponuje celou řadou parametrů a přepínačů, tak jak je většina uživatelů zná z nástrojů příkazových řádek operačních systémů.

Jelikož dotazování probíhá přes MQTT, patří mezi povinné parametry adresa brokeru a téma, které slouží pro dotazování. Mezi nepovinné parametry patří port (v případě, že broker pracuje na jiném než defaultním portu), uživatelské jméno a heslo. Je zde také možnost načíst hodnoty těchto parametrů z konfiguračního souboru. Následně uživatel zvolí příkaz, který bude proveden. Další parametry závisí na daném příkazu a budou popsány níže.

Aby bylo možné zíkat informace o archivovaných datech, je nutné použít klientský program v kombinaci s několika povinnými parametry. Musí být definován název brokeru v rámci systému a téma, z něhož byla data archivována. V některých případech může být pro dané téma archivováno velké množství dat, je tedy možné specifikovat datový rozsah v závislosti na datu. Nad archivovanými daty lze rovněž provádět některé základní operace, tedy nalezení minima a maxima, případně vypočtení průměru nebo sumy. Pokud je nutné některou z operací provádět nad daty, která jsou v databázi uložena v JSON zápisu, je nutné použít ještě parametr, specifikující jméno atributu, obsahující číselné hodnoty.

Data obsažená v odpovědi závisí na typu odeslaného dotazu. Pokud uživatel dotazoval statistiky, obsahuje odpověď data zapsaná v JSON datovém formátu. V tomto případě se s přijatými daty nebude nijak operovat, jednoduše se vypíšou uživateli. Na následujícím příkladu lze vidět, jak je možné získat nejmenší archivované číslo za poslední den.

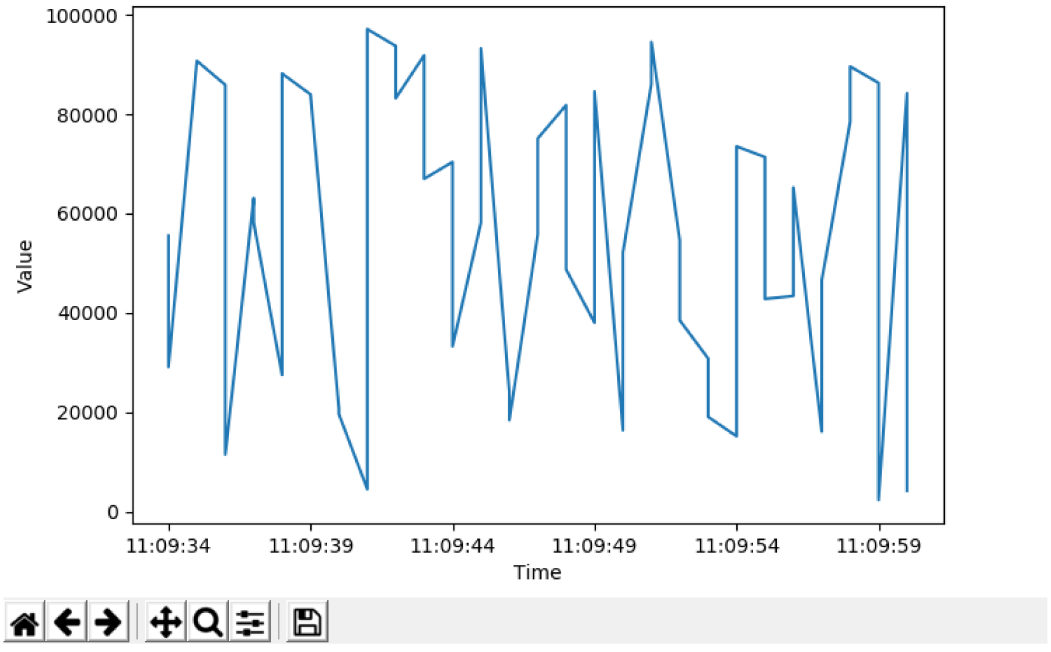
```
C:\python mqttt.py -h localhost -t stats/test get-stats -b brokername
-r topics/rand-num --last 1d --ops min

> Successfully connected to MQTT broker
> Incoming response on request 3e1a8d26-64c2-4f7d-99ce-e387cf5b1073.
> {'result': [2383.0], 'record_type': 'REGEX'}
```

V případě, kdy chce uživatel data vizualizovat pomocí grafu, je proces o něco složitější. Dotaz klientského programu je opět přeložen tak, aby byl validní pro syntaxi dotazovacího jazyka. Zavolá se tedy příkaz `get-graph`, který vrací data jednotlivých os grafu. Tento graf je poté na základě získaných informací zkonstruován. Uživateli se krátce po odeslání dotazu zobrazí vyskakovací okno s vygenerovaným grafem. Okno obsahuje ovládací prvky, které umožňují manipulaci s grafem, přiblížení, oddálení, případně export do souboru.

Graf lze vygenerovat například následujícím příkazem, výsledný graf poté odpovídá tomu z obrázku 11.

```
C:\python mqttt.py -h localhost -t stats/test get-graph -b brokername
-r benchmark/rand-num --last 10d -x "Time" -y "Value"
```

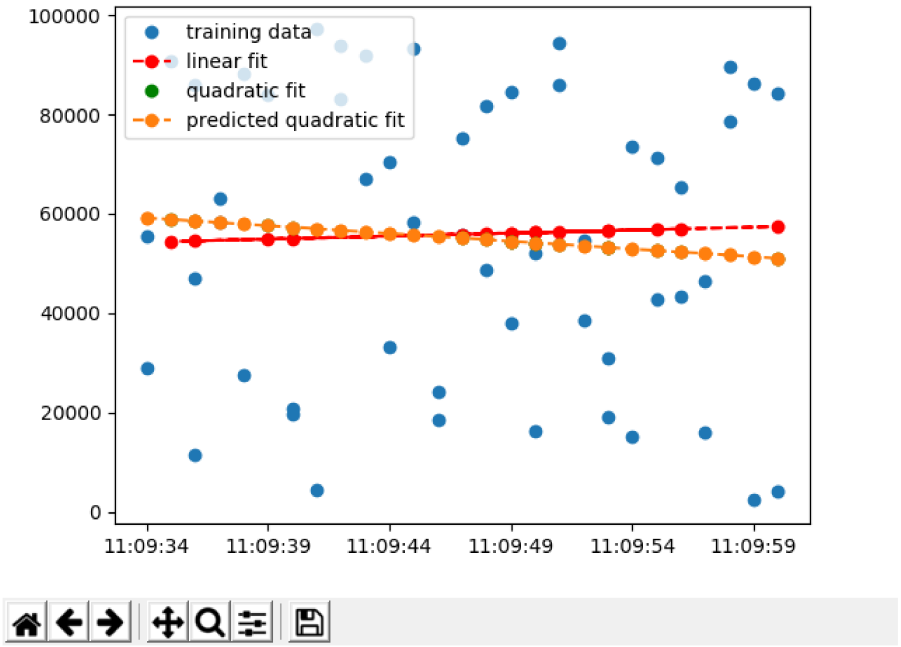


Obrázek 11: Graf vygenerovaný klientským programem

Pokud jde o generování grafů, klientský program požaduje několik povinných parametrů. Uživatel musí zadat název brokeru v rámci aplikace, tedy tak, jak je broker identifikován v databázi. Dalším povinným parametrem je téma, které sloužilo jako zdroj dat pro archivaci.

V případě, že se uživatel snaží vygenerovat graf na základě dat uložených v datovém formátu JSON, je nutno definovat atribut, ve kterém by se číselné hodnoty pro sestrojeni grafu měly nacházet. Pokud uživatel nechce obdržet všechna data z daného brokeru a tématu, měl by použít některý z volitelných parametrů umožňující omezení datového rozsahu na základě data. Rovněž jsou zde k dispozici volitelné parametry pro popis os, případně je zde parametr umožňující export grafu do souboru, kdy graf nebude zobrazen.

V případě generování grafů není možné provádět nad daty operace pro výpočet sumy, minima, maxima a průměru. Lze však vygenerovat graf lineární regrese. Postup je stejný jako v předešlém případě - klientský program přeloží požadavek a aplikace pro sběr dat vrátí osy obsahující data potřebná pro generování grafu. Výsledný graf poté může vypadat jako graf na obrázku 12.



Obrázek 12: Graf lineární regrese vygenerovaný klientským programem

7.2 Implementace programu

Stejně jako v případě programu archivujícího data je i tento program naprogramován v programovacím jazyce Python. Klientská aplikace není stěžejní součástí tohoto systému určeného k archivaci dat, proto bude jeho implementace popsána stručněji.

Program pro vytvoření uživatelského rozhraní používá modul Click. Ten umožnuje vytvořit bohaté uživatelské rozhranní příkazové řádky. Dalším modulem, jež je použit a zároveň není součástí základní instalace Pythonu, je modul Matplotlib, který je používán pro konstrukci grafů. Pro komunikaci s brokerem je opět použit modul Paho MQTT.

Pro potřeby programu byly naimplementovány dva moduly, ze kterých se program skládá. Jedná se o modul MqttManager, který je použit pro odběr a odesílání zpráv, a modul MatplotlibOps, který zajišťuje konstrukci grafů.

7.2.1 MqttManager

Modul MqttManager je jakousi nadvrstvou nad modulem Paho MQTT a zprostředkovává odběr témat a odesílání zpráv. Rovněž obstarává překlad požadavků tak, aby požadavky odpovídaly syntaxi dotazovacího jazyka použitého v programu pro sběr dat.

V okamžiku, kdy uživatel pošle dotaz skrze klientský program, uloží se požadavek do fronty požadavků. Fronta požadavků je implementována slovníkem, jako klíč pro získání informace ze slovníku se používá identifikační řetězec požadavku. Identifikační řetězec se vygeneruje pro každý požadavek a je unikátní. Všechny požadavky jsou v programu představovány objektem Event z modulu threading.

Objekty typu Event zprostředkovávají jednoduchý mechanismus pro komunikaci s vlákny, disponují blokující metodou `wait(timeout)`. Právě tato metoda slouží k čekání na odpověď pro specifický dotaz. V případě, že odpověď přijde, callback metoda prověří, jestli se na odpověď skutečně čeká (proběhne kontrola, jestli je událost s identifikačním řetězcem příchozí zprávy ve slovníku), pokud ano; je nad daným objektem typu Event zavolána metoda `set()`, tím se odblokuje vlákno. Nastane-li situace, že uživatel po odeslání dotazu nezíská odpověď za čas specifikovaný v proměnné `timetout`, bude muset požadavek zopakovat, nebo zkontrolovat, jestli broker funguje korektně.

Jelikož zpracovávání odpovědí probíhá paralalelně v separovaném nerodičovském vlákně, může nastat situace, kdy blokující čekání zamezí zpracovávání dalších dotazů. Právě proto byla implementována již výše zmíněná fronta. Vlákno zpracovávající odpovědi může poté dodatečně zpracovat i zbylé odpovědi tím, že bude obsluhovat frontu.

Přijatá data jsou poté vyextrahována ze zprávy a poslána ke zpracování. Pokud si uživatel přál z přijatých dat vygenerovat graf, je nutno do procesu involvovat modul MatplotlibOps.

Pro lepší ilustraci operací probíhajících v klientské aplikaci, konkrétně v modulu MqttManager, je zde uveden obrázek 13.

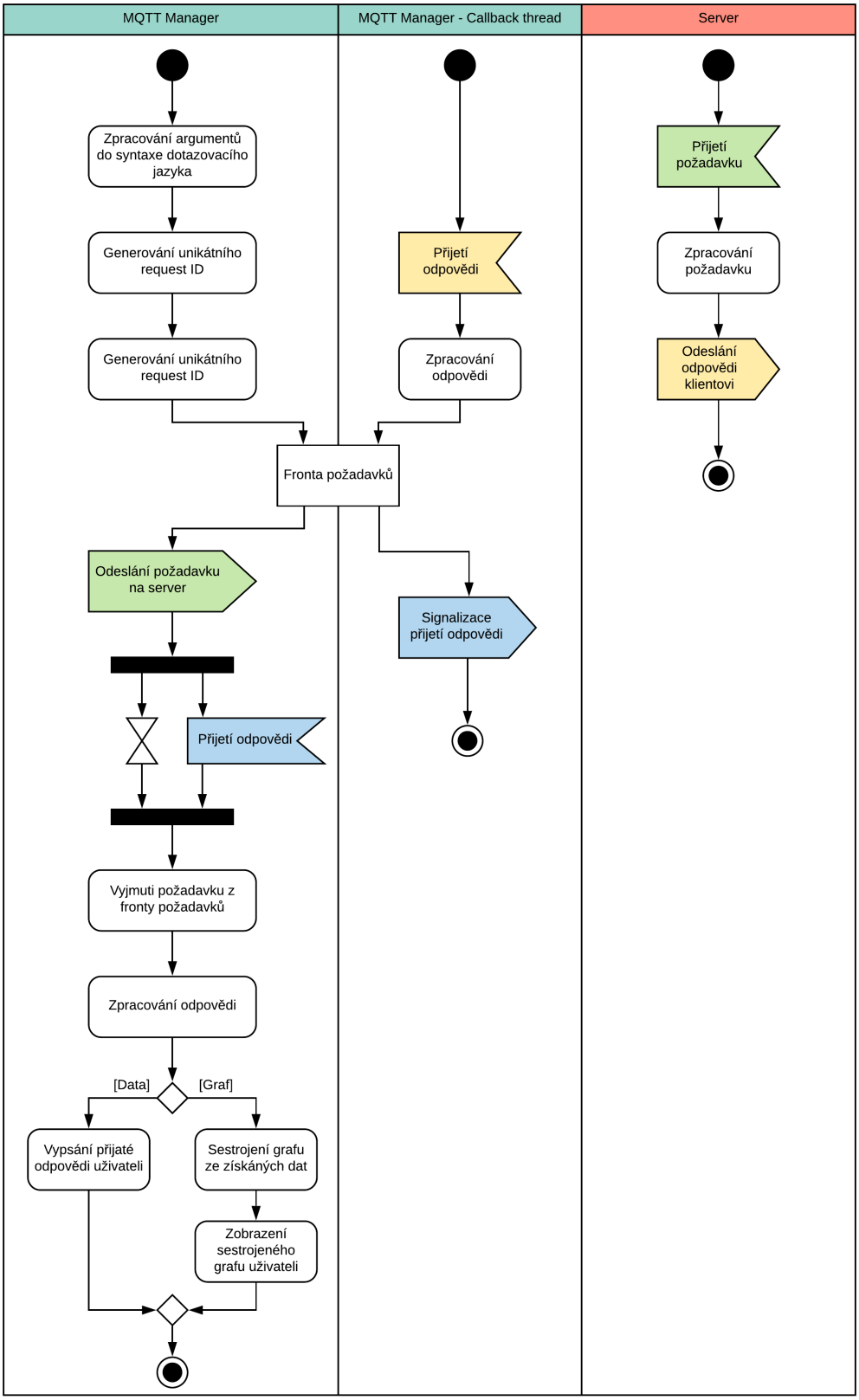
7.2.2 MatplotlibOps

MatplotlibOps je nadvrstvou modulu Matplotlib. Disponuje jak metodami pro konstrukci grafů, tak několika pomocnými metodami.

Je zde metoda, která určí, jestli je v proměnné uloženo číslo. V případech, kdy je graf konstruován z dat typu JSON, je použita právě tato metoda. Důvodem je, že hodnoty jsou v JSONu uloženy bez vazby na původní datový typ a je tedy nutné určit, zdali se jedná o numerické hodnoty, pro validní konstrukci grafu.

Metody pro konstrukce grafů jsou dvě. Jedna slouží pro konstrukci běžného grafu, tedy grafu obsahující hodnoty pro osu x a y. Druhá metoda slouží pro konstrukci grafu lineární regrese. Dvě různé metody jsou zapotřebí, jelikož graf lineární regrese má více os y.

Proces konstrukce grafů probíhá takto: uživatel spustí klientskou aplikaci s vhodnými parametry a očekávaným výstupem je graf. Dotaz je přetlumočen tak, aby byl kompatibilní se syntaxí dotazovacího jazyka použitého v programu pro sběr dat. V případě, že vše funguje korektně, obdrží klientský program data, na základě kterých je možné zkonstruovat graf. Tato data jsou uložena ve slovníku. Slovník obsahuje několik polí, která obsahují data jednotlivých os. Osy jsou ve slovníku identifikovány na základě klíče. Jednotlivé osy jsou tedy poslány modulu MatplotlibOps, do metody zajišťující samotné sestrojení grafu. Nejprve je zjišťěno, zdali se jedná o data typu REGEX nebo JSON. Pokud se jedná o data typu REGEX, není nutno provádět další kroky a data pro osu y jsou připravena k použití. Jedná-li se však o data typu JSON, je nutno na základě názvu atributu, který byl uživatelem definován v parametru, zkontrolovat, zda je v daném atributu číselný údaj, na základě kterého by bylo možné sestrojit graf. Tato data jsou z JSONu vyextrahována a vložena do pole, které představuje osu y. V případě osy x je nutno proiterovat polem obsahující data a převést je na objekty typu datetime. Pole obsahující objekty typu datetime může být následně metodou `date2num` převedeno na pole obsahující Matplotlib datumy, tedy data vhodné pro konstrukci grafů tímto modulem. Těmito poli, obsahujícími patřičná data je naplněn objekt `pyplot`. Následuje popis os, případně export grafu do souboru, v závislosti na zvolených parametrech.



Obrázek 13: Diagram aktivit ilustrující aktivity probíhající v modulu MqttManager

8 Testování programu

Tato část práce bude zaměřena na testování implementované aplikace. Testy, které budou nad aplikací prováděny, budou simulovat běžné používání programu tak, jak se předpokládá, že by ho uživatel používat mohl. Mimo to budou nad aplikací spuštěny testy, které otestují stabilitu a propustnost aplikace, tedy zátěžové testy.

Propustnost aplikace je v tomto textu chápána jako poměr odeslaných dat k archivaci a skutečně archivovanými daty.

8.1 Způsob testování

Pro potřeby testování aplikace byl napsán skript, který je obsažen v řešení. Ten umožňuje odesílání dat ve více procesech, lze také specifikovat hodnotu QoS, interval odesílání zpráv a jejich počet. Odesílaná data jsou náhodně generována a jsou odesílána do čtyřech témat, jedná se o data typu raw, regex a JSON. Data typu regex jsou v jednom případě dále zpracována jako celé číslo, v druhém případě jsou zpracovávána jako kombinace celého čísla a textového řetězce.

Hlavním faktorem systému, který bude testován bude schopnost archivovat příchozí zprávy v určitých intervalech. Skript tedy bude spuštěn s parametry, které odpovídají přenosu 100, 200, 500, 1000, 2000, 5000, 10000 a 20000 zpráv za minutu, v rámci tohoto testování bude odesláno 12000 zpráv k archivaci. Zátěžový test bude proveden s 1, 4, 8 a 16 workery, respektive procesy, odbavující frontu zpráv. Výsledné hodnoty testu budou průměrem ze třech měření.

Ideálním výsledkem testování, je shoda počtu odeslaných zpráv se zprávami archivovanými. Jelikož se zprávy mohou dodatečně archivovat v průběhu odbavování fronty zpráv procesy, bude se počítat množství archivovaných dat v databázi, v okamžiku skončení skriptu.

8.2 Podmínky testování

Pro zátěžové testy byly použity tři stroje. Na jednom byl spuštěn program pro archivaci dat (Collector server), na druhém skript pro zátěžové testování (Benchmark server) a na třetím databázový server. Všechny počítače byly připojeny k síti pomocí ethernetu.

Generování zátěže a sběr zpráv neprobíhal na jednom stroji, jelikož by bylo nežádoucí, aby skript pro zátěžové testování konzumoval hardwarové prostředky aplikaci pro sběr dat nebo obráceně.

8.2.0.1 Benchmark server Jedná se o virtuální stroj běžící na platformě VMware, je osazen osmi jádrovým procesorem Intel Xeon E5-2660, disponující frekvenci 2,20 GHz. Stroj má k dispozici 2 GB operační paměti a 1 GB optickou linku. Operačním systémem je Linux, konkrétně distribuce Debian 8 (jessie). Skript na testování zátěže byl spouštěn pod Pythonem verze 3.4.2.

8.2.0.2 Collector server Collector server je fyzický stroj s operačním systémem Windows 10, rychlost připojení je 120Mbps pro download a 12Mbps pro upload. Zařízení disponuje 4

jádrovým procesorem Intel Core i5-6300HQ s taktem 2,30 GHz a 8 GB operační paměti. Program pro archivaci dat zde běží v Pythonu verze 3.6.0.

8.2.0.3 Databázový server Databáze byla zprovozněna na virtuálním serveru. Server běží na platformě VMware, procesorem je Intel Xeon E5-2650 v3 s taktem 2,30 GHz, v rámci virtualizace je ale jednoprocesorový. Stroj má 1 GB operační paměti a SSD úložiště. Server je napojen na optickou 1GB linku. Server je poháněn operačním systémem Ubuntu 16.04, na něm je pak spuštěno MySQL ve verzi 5.7.21.

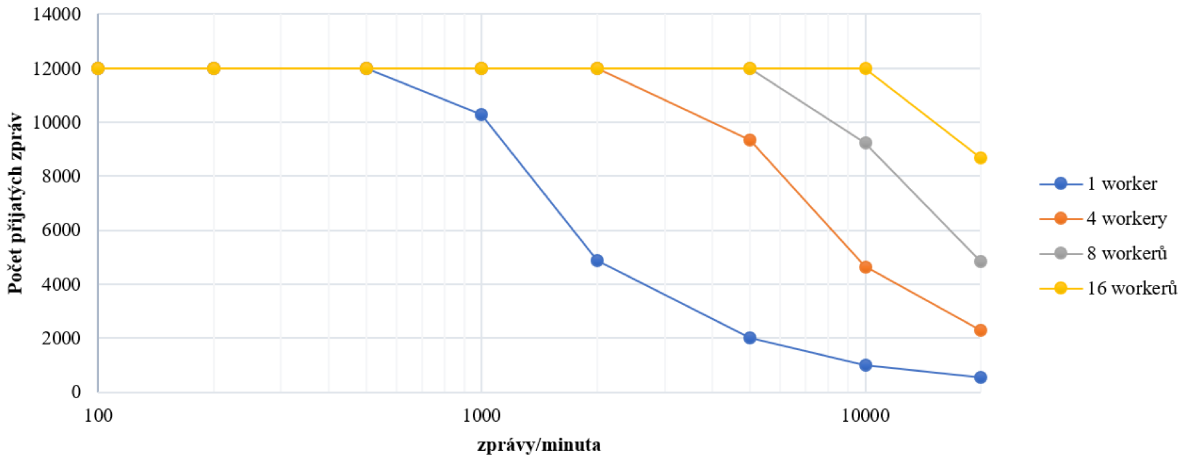
8.3 Výsledky testování

Výsledky zátěžového testování je možné vidět v tabulce 2, ty jsou poté vizualizovány grafem na obrázku 14. Tabulka obsahuje množství zpráv, archivovaných v databázi v okamžiku skončení skriptu pro testování zátěže. Naměřená data reprezentují schopnost aplikace archivovat zprávy poslané ve velice nízkém intervalů z několika zdrojů.

Z uvedené tabulky 2 vyplývá, že platí přímá úměra mezi počtem procesů a schopnosti přijímat zprávy. Použití většího množství procesů, obsluhující frontu zpráv, má za následek větší propustnost příchozích zpráv.

Počet workerů	**Rychlost odesílání zpráv [zprávy/minuta]**							
	100	**200**	**500**	**1000**	**2000**	**5000**	**10000**	**20000**
1	12000	12000	12000	10288	4865	2013	990	554
4	12000	12000	12000	12000	12000	9337	4643	2286
8	12000	12000	12000	12000	12000	12000	9221	4818
16	12000	12000	12000	12000	12000	12000	12000	8661

Tabulka 2: Výsledky zátěžového testu programu pro archivaci dat



Obrázek 14: Graf vizualizující tabulku 2

8.4 Závěr testování

Z provedených testů vyplývá, že použitím jednoho workeru je možné přijímat zprávy rychlostí přibližně pět set zpráv za minutu. Takové propustnosti zprvu dosahovala aplikace, jelikož se nepočítalo s potřebou víceprocesního zpracování přijatých zpráv.

Zpráva je po přijetí většinou ještě dále zpracovávána, a ačkoliv se jedná o operace, které trvají několik málo milisekund, velké množství zpráv zapříčiní, že proces nestíhá zpracovávat příchozí zprávy a dochází k jejich zahazování, což zásadně ovlivňuje propustnost.

Právě z tohoto důvodu byly do aplikace doimplementovány worker pool a fronta zpráv, to umožnilo zpracování více příchozích zpráv současně a jejich archivaci.

Výkon a propustnost programu je tedy závislý na hardwarových prostředcích hostujícího zařízení. Fronta přijatých zpráv je implementována pomocí objektu Queue (fronta). Ten představuje frontu zpráv, jejíž velikost je nastavitelná a může být teoreticky nekonečně velká. Fronta zpráv je dále odbavována předem nastaveným počtem procesů; ten se odvíjí od konkrétního použitého procesoru. V rámci provedených testů nebyla fronta nijak limitována.

Protože program disponuje víceprocesním zpracováním přijatých zpráv, umožnuje archivaci přibližně deset tisíc zpráv za minutu, což je 167 zpráv za vteřinu, za použití šestnácti procesů odbavujících frontu zpráv. To by se mohlo lišit v případě použití jiného hardwaru. Z celkového testu vyplývá, že zvýšení počtů procesů zvýší také propustnost dat.

Pokud bude fronta příliš malá, počet procesů bude nízký a frekvence přijímaných zpráv, které budou určeny k archivaci příliš vysoká, procesy nebudou schopny odbavovat frontu; ta se po čase přeplní a dojde ke ztrátě zpráv. Uživatel tedy musí korektně nastavit aplikaci na základě hardwarových možností a daných okolností, případně zvolit nejlepší kompromis.

9 Alternativní řešení

Tato část práce bude zaměřena na alternativní systémy, umožňující archivaci dat pomocí protokolu MQTT.

Díky rozmachu IoT vznikla spousta služeb - například ThingSpeak, ThingBoard, Cayenne nebo AdafruitIO. Všechny tyto platformy umožnují sběr, export a vizualizaci dat. Odlišnosti jsou zejména v uživatelském rozhraní, cenové politice produktu nebo v lehce odlišných implementacích sběru dat pomocí protokolu MQTT.

Rovněž bylo vyvinuto několik utilit a programů umožňujících archivaci dat pomoci MQTT protokolu v rámci komunitního vývoje. V této kapitole bude několik takových služeb a programů představeno.

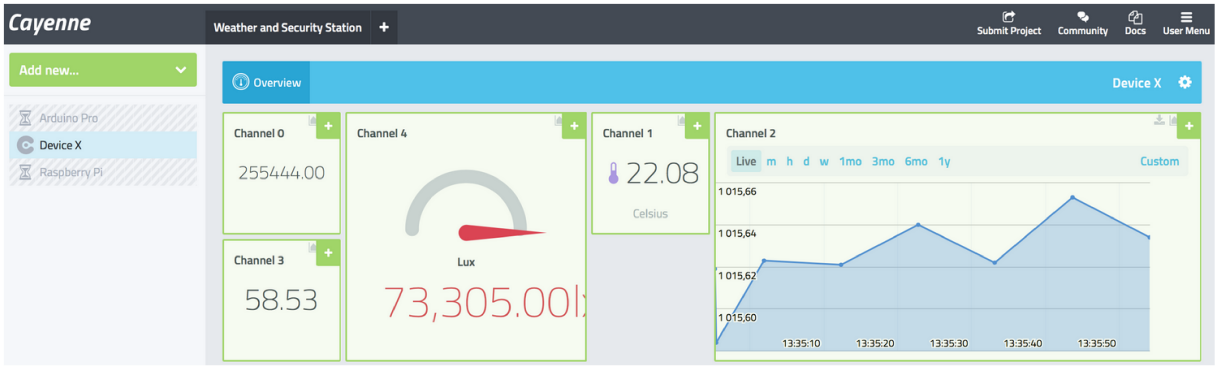
9.1 Cayenne

Cayenne je příkladem komplexní internetové služby pro archivaci, vizualizaci a export dat. Umožňuje uživatelům vytvořit vlastní cloudové úložiště pro IoT data. Podporuje celou řadu zařízení, která se běžně používají v IoT, jmenovitě například Raspberry Pi, Arduino a rovněž velké množství LoRaWAN zařízení.

Důvodem, kvůli kterému je ale Cayenne zmíněn v tomto textu, je jeho MQTT API. Díky němu je možné sbírat jakákoliv data a ta poté vizualizovat pomocí velkého mnžostvím widgetů a grafů, kterými Cayenne disponuje. Rovněž je možné odeslat data z cloudu směrem k zařízením, a tím ovládat IoT zařízení vzdáleně.

Pokud chce uživatel přidat zařízení do Cayenne, musí v uživatelském rozhraní zvolit možnost vytvořit nové zařízení. Zde je možné zvolit z několika předvoleb nebo vytvořit zařízení nové. Pokud se uživatel rozhodne pro MQTT protokol, jsou mu vygenerovány přihlašovací údaje pro Cayenne broker. Je rovněž doporučeno použít některé z Cayenne SDK, díky němu je totiž možné odesílat zprávy v takovém formátu, v jakém je Cayenne IoT cloud očekává.

Cayenne umožňuje vytvoření dashboardu pro každé zařízení, jak je možné vidět na obrázku 15. Rovněž disponuje aplikací pro mobilní telefony s operačním systémem Android nebo iOS, takže je možné k archivovaným datům přistupovat odkudkoliv.



Obrázek 15: Cayenne dashboard pro data přijatá protokolem MQTT

9.2 influx4mqtt

Jednoduchá utilita naprogramovaná v JavaScriptu, sloužící pro archivaci dat pomocí MQTT protokolu a jejich následné uložení do databáze. Jedná se o databázi InfluxDB, která je velice vhodná pro time-series data, tedy data závislá na časové informaci [12].

Utilitu influx4mqtt je velice jednoduché zprovoznit a používat, stačí pomocí balíčkovacího manažeru NPM pro JavaScript nainstalovat influx4mqtt. Spuštění pak probíhá tak, že se program spustí s určitými parametry definující téma, adresu brokeru, adresu a přihlašovací údaje k InfluxDB.

Pokud tedy uživatel chce odebírat několik různých témat, která nelze odebírat pomocí wildcard, je nutno spustit několik instancí tohoto programu.

Pokud by byla porovnána aplikace, vyvinutá v rámci této práce s utilitou influx4mqtt, rozdíly by bylo možné vidět v typu použité databáze, kdy utilita influx4mqtt využívá databázi InfluxDB, která je pro IoT data vhodnější než běžné relační databáze. Na druhou stranu ale utilita neumoňuje uživateli definovat, jaká data budou archivována, dále pak utilita nedisponuje možností získání archivovaných dat, případně provedením jednoduché statistické operace nad archivovanými daty.

9.3 mqtt2graphite

Jedná se o utilitu sloužící k archivaci a vizualizaci dat, přijatých protokolem MQTT naprogramovanou v jazyce Python, využívající MQTT klientskou knihovnu Paho.

Program odebírá libovolné množství MQTT témat, poté extrahuje hodnoty z příchozích zpráv a následně data odešle do monitorovacího nástroje Graphite.

Graphite se skládá ze dvou komponent - webového frontendu a backendu Carbon, který představuje datové úložiště. Data jsou tedy posílána na Carbon server, ten je zpřístupňuje pro real-time vizualizaci a poté, hned jak je to možné, je uloží na disk. Data mohou být vizualizována ještě před samotným uložením na disk [14].

Hodnoty obsažené v příchozích zprávách mohou být číselné, nebo zapsány ve formátu JSON. Pokud budou data zapsána ve formátu JSON, dojde k vyextrahování všech klíčů a hodnot. V okamžiku, kdy program ze zpráv vyextrahuje data, odešle je na Carbon server prostřednictvím UDP socketu [15]. Odtud jsou pak načteny softwarem Graphite a vizualizovány, výslednou vizualizaci je možné vidět na obrázku 16.

Obrázek 16: Graf vygenerovaný nástrojem Graphite na základě dat z MQTT [15]

10 Závěr

Cílem této práce bylo seznámit čtenáře s technologií MQTT a následně navrhnout a popsat implementaci programu, jehož úkolem je archivovat data, přenášená prostřednictvím protokolu MQTT. Nakonec byla testována výkonnost, respektive propustnost programu. Výstupem testování jsou data, na základě kterých si čtenář může udělat představu o výkonnosti programu.

Výsledkem práce je tedy program, pro archivaci dat, přenášených protokolem MQTT. Pro jeho implementaci byl zvolen programovací jazyk Python, jelikož umožnuje rychlý, dynamický vývoj a disponuje velkým množstvím rozšiřujících knihoven, které urychlily implementaci systému.

Během návrhu bylo nutno vymyslet způsob archivace dat. Výsledkem tohoto snažení bylo zavedení zjednodušených regulárních výrazů, umožňující definovat konkrétní formát příchozí zprávy a tu poté archivovat.

Program data archivuje do MySQL databáze, ale vzhledem k použitému ORM je případně možné použít i jiný relační databázový systém.

Jedním z požadavků na program byla implementace vhodného rozhraní pro přístup k archivovaným datům. Navrhl jsem a následně vyvinul několik způsobů, jak k datům přistupovat. Primárním způsobem je využití dotazovacího jazyku, kdy se příkaz odešle prostřednictvím MQTT zprávy do tématu, určeného pro požadavky. Odpověď je poté odeslána do nadtématu určeného pro odpovědi. Dalším způsobem, jak přistupovat k nasbíraným datům je použití klientské aplikace, která je součástí řešení. Ta rovněž využívá dotazovacího jazyka, stejně jako předchozí možnost, nabízí však příjemnější uživatelské rozhraní a umožňuje také vizualizaci dat, pomocí grafů. Poslední možností pro přístup k archivovaným datům je použití REST API.

Výsledný systém také umožňuje provádění základních operací nad nasbíranými daty. Jedná se o jednoduché operace na databázové úrovni, tedy nalezení minima a maxima, výpočet průměru nebo sumy. Aplikace však disponuje i pokročilejšími operacemi na aplikační úrovni, jedná se o lineární regresi nebo predikci hodnot.

Pro otestování správné funkčnosti a propustnosti byly provedeny zátěžové testy, z těch vyplynulo, že pro zvládnutí archivace většího množství příchozích zpráv, je nutné využívat víceprocesní zpracovávání fronty zpráv. Aplikaci jsem modifikoval, tak aby uživateli umožnila nastavit, kolik procesů bude frontu obsluhovat. Ukázalo se, že zpracovávání fronty jedním procesem, umožňuje odběr přibližně 500 zpráv za minutu. Obsluha fronty zpráv 16 procesy umožňuje odběr přibližně 10000 zpráv za minutu. Systém netrpí problémy se stabilitou a při korektním nastavení pro specifické hardwarové vybavením, nedocházelo ke ztrátě dat před archivací.

Výsledný program, který jsem vyvinul v rámci této práce, splňuje veškeré zadané požadavky. Předpokládám, že by systém mohl dosahovat optimálnějšího výkonu, kdyby byl použít jiný programovací jazyk, například C nebo C++. Pro sběr velkého množství dat, by rovněž mohlo být optimálnější použít některou z time-series databázi, například InfluxDB. Time-series databáze nebyla použita, jelikož jsem v době implementace systému nedisponoval dostatkem znalostí

a zkušeností s tímto typem databází. Rovněž jsem v systému použil ORM framework, který podporuje pouze relační databáze.

Pokud srovnám systém vyvinutý v rámci této práce, s některými systémy podobného charakteru, které byly vyvinuty v rámci komunitního vývoje, vynikl by tento systém především ve své multifunkcionalitě. Systém obsahuje jak prostředky pro archivaci, tak pro vizualizaci, také umožňuje provádět operace nad archivovanými daty. Rovněž možnost definovat konkrétní formát zprávy, určenou k archivaci je unikátní a může být velice užitečná.

Tato práce reaguje na současnou popularitu a rozvoj technologií v oblasti internetu věcí, kde se protokol MQTT stal velmi populární pro komunikaci mezi zařízeními. V této práci jsem navrhl systém pro efektivní uchovávání zpráv přenášených tímto protokolem, s ohledem na univerzální použití a možnou rozšiřitelnost.

Literatura

[1] Martin Malý. *Protokol MQTT: komunikační standard pro IoT* [online]. Copyright © Root.cz [cit. 24.04.2018]. Dostupné z: https://www.root.cz/clanky/protokol-mqtt-komunikacni-standard-pro-iot/

[2] IBM Corporation. *IBM Knowledge Center - Configuring message hubs* [online]. Copyright © 2017, IBM Corp. [cit. 24.04.2018]. Dostupné z: https://www.ibm.com/support/knowledgecenter/SSCGGQ_1.2.0/com.ibm.ism.doc/Administering/ad00360_.html

[3] Martin Malý. *MQTT – co, jak, kde?* [online]. [cit. 24.04.2018].

Dostupné z: https://iotta.cz/mqtt-co-jak-kde/

[4] dc-square GmbH. Introducing the MQTT Security Fundamentals [online]. Copyright © 2018, dc-square GmbH [cit. 24.04.2018]. Dostupné z: https://www.hivemq.com/blog/introducing-the-mqtt-security-fundamentals

[5] Roger Light. Mosquitto dokumentace [online]. [cit. 24.04.2018]. Dostupné z: https://mosquitto.org/man/mosquitto-conf-5.html

[6] Toby Jaffey. *MQTT and CoAP, IoT Protocols* [online]. Copyright © Eclipse Foundation [cit. 24.04.2018]. Dostupné z: https://www.eclipse.org/community/eclipse_newsletter/2014/february/article2.php

[7] Ben Mesander. *Using UDP in Internet-of-Things Devices* [online]. Copyright © 2018 Cardinal Peak [cit. 24.04.2018]. Dostupné z: https://cardinalpeak.com/blog/using-udp-in-internet-of-things-devices/

[8] John Carbone. *Speed up machine-to-machine networking with UDP* [online]. Copyright © 2018 AspenCore [cit. 24.04.2018]. Dostupné z: https://www.embedded.com/design/real-world-applications/4426378/Speed-up-machine-to-machine-networking-with-UDP

[9] David Barnett. *MQTT and DDS for M2M: Disparate Approaches to the Internet of Things* [online]. Copyright © 2007-2017, Real-Time Innovations [cit. 24.04.2018]. Dostupné z: https://www.rti.com/blog/2013/05/08/mqtt-dds-m2m-protocol-internet-of-things/

[10] David Barnett. *Comparison of MQTT and DDS as M2M Protocols for the Internet of Things* [online]. Copyright © 2007-2013, Real-Time Innovations [cit. 24.04.2018]. Dostupné z: https://www.slideshare.net/RealTimeInnovations/comparison-of-mqtt-and-dds-as-m2m-protocols-for-the-internet-of-things

[11] Oracle Corporation and/or its affiliates. *MySQL 5.7 Reference Manual :: 11.6 The JSON Data Type. MySQL* [online]. Copyright © 2018, Oracle Corporation and/or its affiliates [cit. 24.04.2018]. Dostupné z: https://dev.mysql.com/doc/refman/5.7/en/json.html

[12] Sebastian Raff. *hobbyquaker/influx4mqtt: Insert incoming MQTT values into InfluxDB* [online]. [cit. 24.04.2018]. Dostupné z: https://github.com/hobbyquaker/influx4mqtt

[13] Rei Vilo. *IoT with Cayenne MQTT* [online]. Copyright © 2010-2018 , Rei Vilo [cit. 24.04.2018]. Dostupné z: https://embeddedcomputing.weebly.com/iot-with-cayenne-mqtt.html

[14] *Carbon - Graphite* [online]. [cit. 24.04.2018].

Dostupné z: http://graphite.wikidot.com/carbon

[15] Jan-Piet Mens. *jpmens/mqtt2graphite: Subscribe to MQTT topics and push to Graphite's Carbon server* [online]. [cit. 24.04.2018].

Dostupné z: https://github.com/jpmens/mqtt2graphite

A Zdrojové kódy na CD

Přiložené CD obsahuje zdrojové kódy programu pro archivaci dat, klientské aplikace a také skriptu použitého pro zátěžové testy systému.